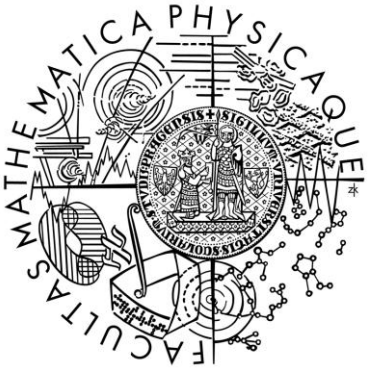
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Bc. Peter Szépe

Vícenásobná podobnost RNA struktur

Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. David Hoksza, Ph.D.

Studijní program: Informatika

Studijní obor: Softwarové inženýrství

Praha 2013

Chtěl bych poděkovat mému vedoucímu RNDr. Davidu Hokszovi, Ph.D. za jeho čas, podnětné rady a zapůjčenou literaturu. Také děkuji bratrovi, Štefanovi Szépemu, za pomoc s anglickým jazykem a své rodině a přátelům za podporu.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 12. dubna 2013 Peter Szépe

Název práce: Vícenásobná podobnost RNA struktur

Autor: Bc. Peter Szépe

Katedra / Ústav: Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. David Hoksza, Ph.D., Katedra softwarového inženýrství

Abstrakt: Práce vychází z algoritmu SETTER (SEcondary sTructure-based TERtiary Structure Similarity Algorithm), což je určen k porovnání 3D struktur RNA. SETTER v původní verzi umí porovnávat pouze dvojice RNA, nicméně mnoho reálných aplikací vyžaduje přiřazení podobnosti n-tici RNA struktur. Hlavní myšlenkou MultiSETTERu je dobře známý postup používaná pro vícenásobné sekvenční zarovnání, což je založen na metodě Neigbour-Joining – metoda pro kalkulaci taxonomického stromu ze vzdálenosti mezi taxony – a na zarovnání dvojic. V každém kroku zarovnáme nejbližší dvojici podle taxonomického stromu. Pro dosažení dobrého výsledku bylo nezbytné vymyslet metodu, která vytvoří takzvanou průměrnou RNA strukturu z dvou RNA, nesoucí v sobě charakteristické vlastnosti obě struktur.

Klíčová slova: RNA, 3D struktura, podobnost

Title: Multiple RNA Structure Similarity

Author: Bc. Peter Szépe

Department: Department of Software Engineering

Supervisor: RNDr. David Hoksza, Ph.D., Department of Software Engineering

Abstract: The work is based on the algorithm SETTER (Secondary Structure Tertiary Structure-based Similarity Algorithm), which is designed to compare the 3D structures of RNA. SETTER in the original version can only compare pairs of RNA, however many applications require real similarity comparison between a set of RNA structures. The main idea used in MultiSETTER is a well-known approach used for multiple sequence alignment, which is based on the Neigbour-Joining method – a method for calculation of the taxonomic tree from distances between taxa – and on pairwise alignment. At each step the closest pair is aligned according to the taxonomic tree. To achieve good results, it was necessary to invent a method that creates a fictive average RNA structure by merging two RNAs that shares the structural characteristics of both RNA.

Keywords: RNA, 3D structure, similarity

Contents

1 Introduction

In this work a novel multiple structural alignment method for RNA structures will be introduced based on the SETTER (Secondary Structure Tertiary Structure-based Similarity Algorithm) algorithm, which is designed to compare the three dimensional structures of two RNAs. There are few multiple structural alignment tools, but there is no other available software tool, which works with three dimensional structures.

Why are RNAs interesting? There is a family of RNAs, the non-coding RNAs, that seem to comprise a hidden layer of internal signals that control various levels of gene expression in physiology and development, including chromatin architecture/epigenetic memory, transcription, RNA splicing, editing, translation and turnover. RNA regulatory networks may determine most of our complex characteristics that play a significant role in disease and constitute an unexplored world of genetic variation both within and between species.

Why is it important to align three dimensional structures? Since the function of RNA is more correlated to the three dimensional layout, then the order of nucleotide bases, to properly understand the function of RNA molecules requires methods for comparing and analyzing their 3D structures.

2 Biological and Bioinformatics Background

2.1 Introduction to DNA and RNA

Deoxyribonucleic acid (DNA) and Ribonucleic acid (RNA) are long linear polymers called nucleic acids that carry information in a form that can be passed from one generation to the next; respectively DNA stores the information and RNA is used to transfer the information. These macromolecules consist of a large number of linked nucleotides; each composed of a sugar, a phosphate, and a base. Sugars linked by phosphates form the common backbone.

There are four kinds of bases: adenine (A), guanine (G), cytosine (C) and thymine (T, DNA only) or uracil (U, RNA only). Genetic information is stored in the sequence of bases along a nucleic acid chain.

Although RNA probably functioned as the genetic material very early in evolutionary history, the genes of all modern cells and many viruses are made of DNA.

(Berg, et al., 2002 p. 193)

2.1.1 Ribonucleic Acid (RNA)

Ribonucleic acid (RNA) is one of the three major macromolecules (along with DNA and proteins) that are essential for all known forms of life.

The ribonucleic acids can be divided into multiple groups:

- Messenger RNA (mRNA) molecules are the information-carrying intermediates in protein synthesis. The genes specify the kinds of proteins that are made by cells, but DNA is not the direct template for protein synthesis. Messenger RNA is the template for protein synthesis or translation. (Berg, et al., 2002 p. 193 and 214)
- Transfer RNA (tRNA) carries amino acids in an activated form to the ribosome for peptide-bond formation, in a sequence dictated by the mRNA template. Transfer RNA consists of about 75 nucleotides, which makes it the smallest of the RNA molecules. (Berg, et al., 2002 p. 214)

- Ribosomal RNA (rRNA) is the RNA component of the ribosome. Plays both a catalytic and a structural role in protein synthesis - it comprises the predominant material within the ribosome, which is about 60% rRNA and 40% protein by weight.
- Non-coding RNAs – all RNAs except of messenger RNAs are non-coding RNAs. The function of these RNAs could be involved in gene regulation, RNA processing and other roles. Certain RNAs are able to catalyze chemical reactions such as cutting and ligating other RNA molecules and the catalysis of peptide bond formation in the ribosome.

2.1.2 The RNA World Hypothesis

The RNA world hypothesis proposes that evolution based on RNA replication preceded the appearance of protein synthesis. It was suggested that catalysts made entirely of RNA were likely to be important at this early stage in the origins of life, but the possibility that RNA catalysts might still be present in contemporary organisms was overlooked. The discovery of ribozymes initiated extensive discussion of the role of RNA in the origins of life.

The RNA world means different things to different authors, so it would be futile to attempt to give a restrictive definition. All RNA world hypotheses include three basic assumptions: (1) At some time in the evolution of life, genetic continuity was assured by the replication of RNA; (2) Watson-Crick base-pairing was the key to replication; (3) genetically encoded proteins were not involved as catalysts.

(Joyce, et al., 1993)

2.1.3 Interesting Ribonucleic Acids

In this section some interesting RNA families will be introduced to justify the importance of research in the field of ribonucleic acids.

2.1.3.1 Viruses

The genes in all cellular organisms are made of DNA; which is also true for some viruses, however their genetic material also might be RNA. Viruses are genetic elements enclosed

in protein coats that can move from one cell to another but are not capable of independent growth.

An important class of RNA virus comprises the retroviruses, called that way because the genetic information flows from RNA to DNA rather than from DNA to RNA. This class includes human immunodeficiency virus 1 (HIV-1), the cause of AIDS, as well as a number of RNA viruses that produce tumors in susceptible animals. Retrovirus particles

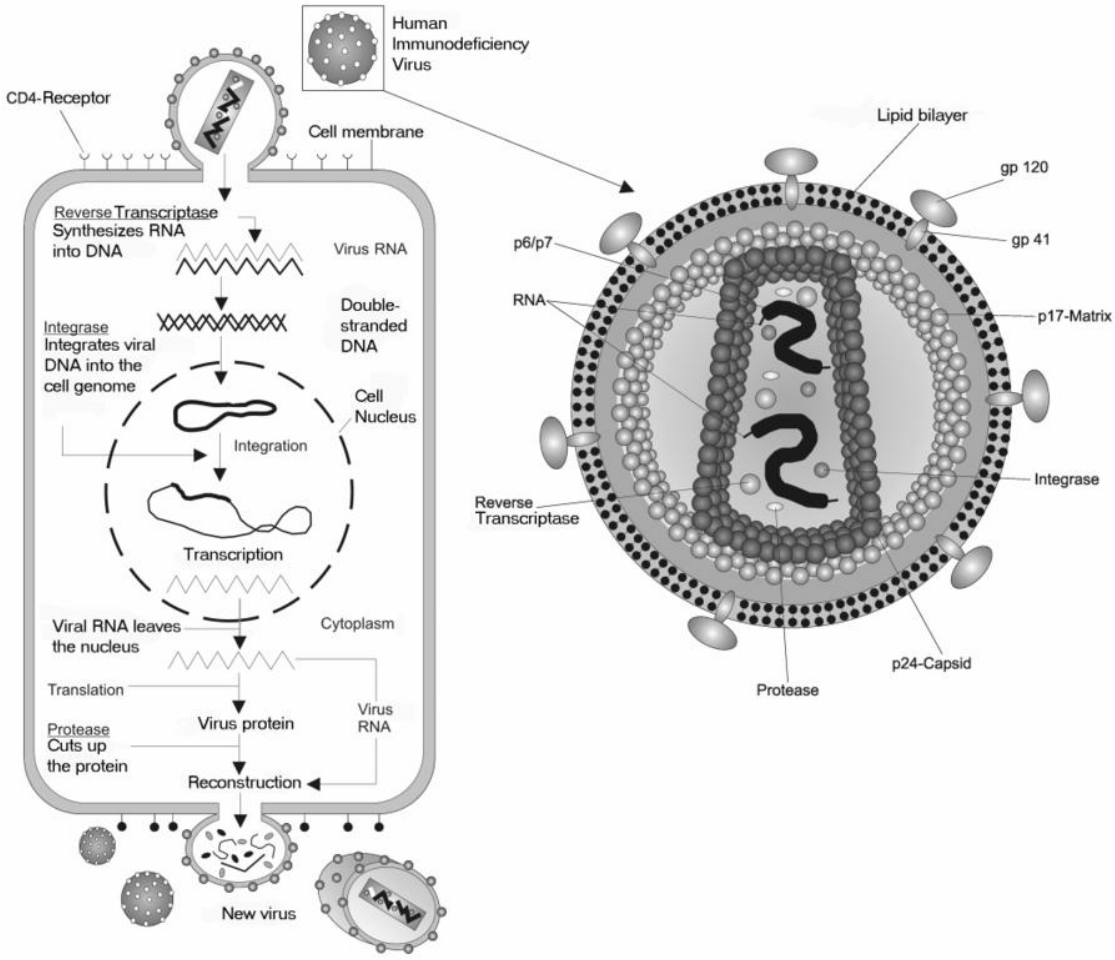


Figure 1: HIV-1 Retrovirus; *http://en.wikipedia.org/wiki/File:Hiv_gross.png*

contain two copies of a single-stranded RNA molecule. On entering the cell, the RNA is copied into DNA through the action of a viral enzyme called reverse transcriptase. The resulting double-helical DNA version of the viral genome can become incorporated into the chromosomal DNA of the host and is replicated along with the normal cellular DNA.

At a later time, the integrated viral genome is expressed to form viral RNA and viral proteins, which assemble into new virus particles. (Berg, et al., 2002 p. 212)

2.1.3.2 Non Coding RNA

The term non-coding RNA (ncRNA) is commonly employed for RNA that does not encode a protein; which does not mean that such RNAs do not contain information nor have function. Although it has been generally assumed that most genetic information is transacted by proteins, recent evidence suggests that the majority of the genomes of mammals and other complex organisms is in fact transcribed into ncRNAs, most of which have unknown functions. These RNAs seem to comprise a hidden layer of internal signals that control various levels of gene expression in physiology and development, including chromatin architecture/epigenetic memory, transcription, RNA splicing, editing, translation and turnover. RNA regulatory networks may determine most of our complex characteristics that play a significant role in disease and constitute an unexplored world of genetic variation both within and between species. (Human Molecular Genetics, 2006, Vol. 15, Review Issue 1, 2006)

2.2 Building Blocks of Nucleic Acids

As already mentioned, nucleic acids are linear polymers built up from similar units. Each monomer unit within the polymer consists of three components: sugar, phosphate, and a base. The sugar components in nucleic acids are linked to each other by phosphodiester bridges referred to as the backbone of the nucleic acid (Figure 2: The RNA backbone), In case of RNA, the sugar units are riboses which are a monosaccharaides containing five carbon atoms. The backbone of RNA has always the same structure but the bases vary from one monomer to the next. Adenine (A) and guanine (G) bases are derivatives of purine while cytosine (C) and thymine (T, DNA only) or uracil (U, RNA only) are derivatives of pyrimidine (Figure 3: The nucleotide bases)

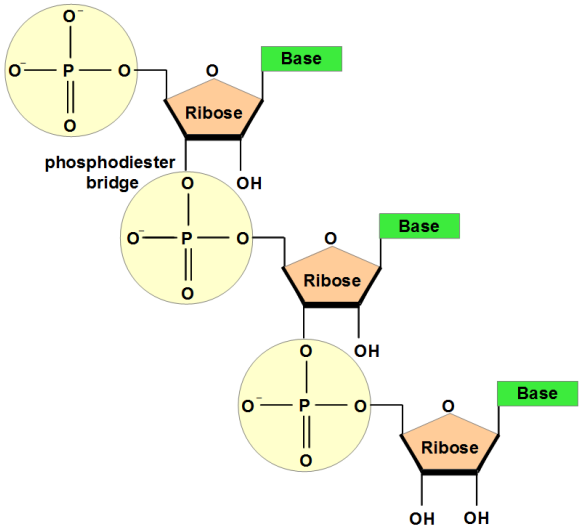


Figure 2: The RNA backbone; the sugar components – riboses – in RNA are linked to each other by phosphodiester bridges.

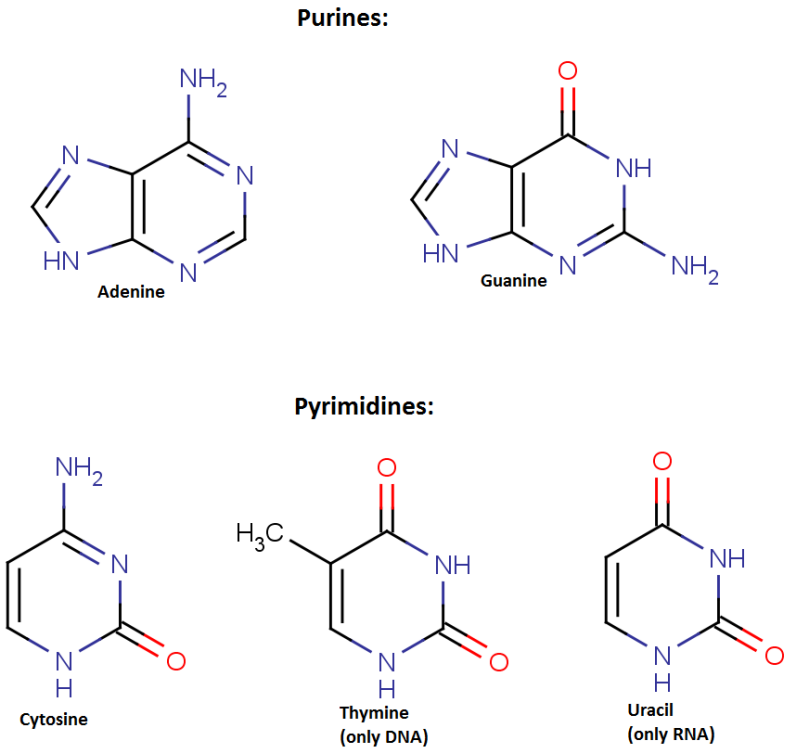


Figure 3: The nucleotide bases

The bases have an additional special property: they form specific pairs with one another that are stabilized by hydrogen bonds. These pairs are called Watson-Crick base pairs. In case of RNAs, the Watson-Crick base pairs are G • C and A • U. RNA structure can contain a wide variety of non-Watson-Crick interactions (e.g. G • U or A • A).

2.2.1 Folding

Single-stranded nucleic acids, like RNA, often fold back on themselves to form well-defined structures/motifs. This is in contrast to DNA structures which come double-stranded allowing them to form the famous double helix. The simplest and most common structural motif formed is a stem-loop, created when two complementary sequences within a single strand come together to form double-helical structures ends in a short unpaired loop. The first prerequisite is a presence of a sequence that can fold back on itself, therefore in many cases, these double helices are made up entirely of Watson-Crick base pairs. In some cases the structures include mismatched or unmatched (bulged) bases (see Figure 5: Secondary Structure Motifs). Such mismatches destabilize the local structure but introduce deviations from the standard double-helical structure that can be important for higher-order folding and for function.

2.2.2 The Structure of Nucleic Acids

The RNA structures can be defined on different levels of structural detail. In the following sections, the given types of nucleic acid representation will be demonstrated on RNA having Protein Data Bank identifier 1QRS with chain identifier B (see 6.1.2).

2.2.2.1 Primary Structure:

The first level of organization is the sequence of bases attached to the sugar–phosphate backbone, thus the primary structure is a linear sequence of nucleotides which determines the coded information. Specifically, every triplet of the nucleotide chain corresponds to given amino acid forming so called genetic code.

```
UGG GGU AUC GCC AAG
CGG UAA GGC ACC GGA
UUC UGA UUC CGG CAU
UCC GAG GUU CGA AUC
CUC GUA CCC CAG CCA
```

Figure 4: The primary structure of 1QRS

2.2.2.2 Secondary Structure:

In salty water, the RNA molecules fold back on themselves via bonds between Watson–Crick complements (A • U, C • G) or wobble pairs (G • U, or G • T) leading to double-stranded helices interrupted by single-stranded regions in internal loops or hairpin loops. The enumeration of the base-paired regions or helices constitutes a description of the second level of organization, the secondary structure, thus it may be described as a graph with connections between paired bases on a polymer backbone.

The methods available to deduce the secondary structure of an RNA molecule are mainly of three types: the phylogenetic approach, the theoretical prediction, and chemical/enzymatic methods. (Paradigms for computational nucleic acid design, 2004), (Westhof, et al., 2000)

2.2.2.2.1 Secondary Structure Motifs

While DNA mostly exist as fully base paired double helixes, RNA tends to fold back on them and often forms complicated base-pairing interactions.

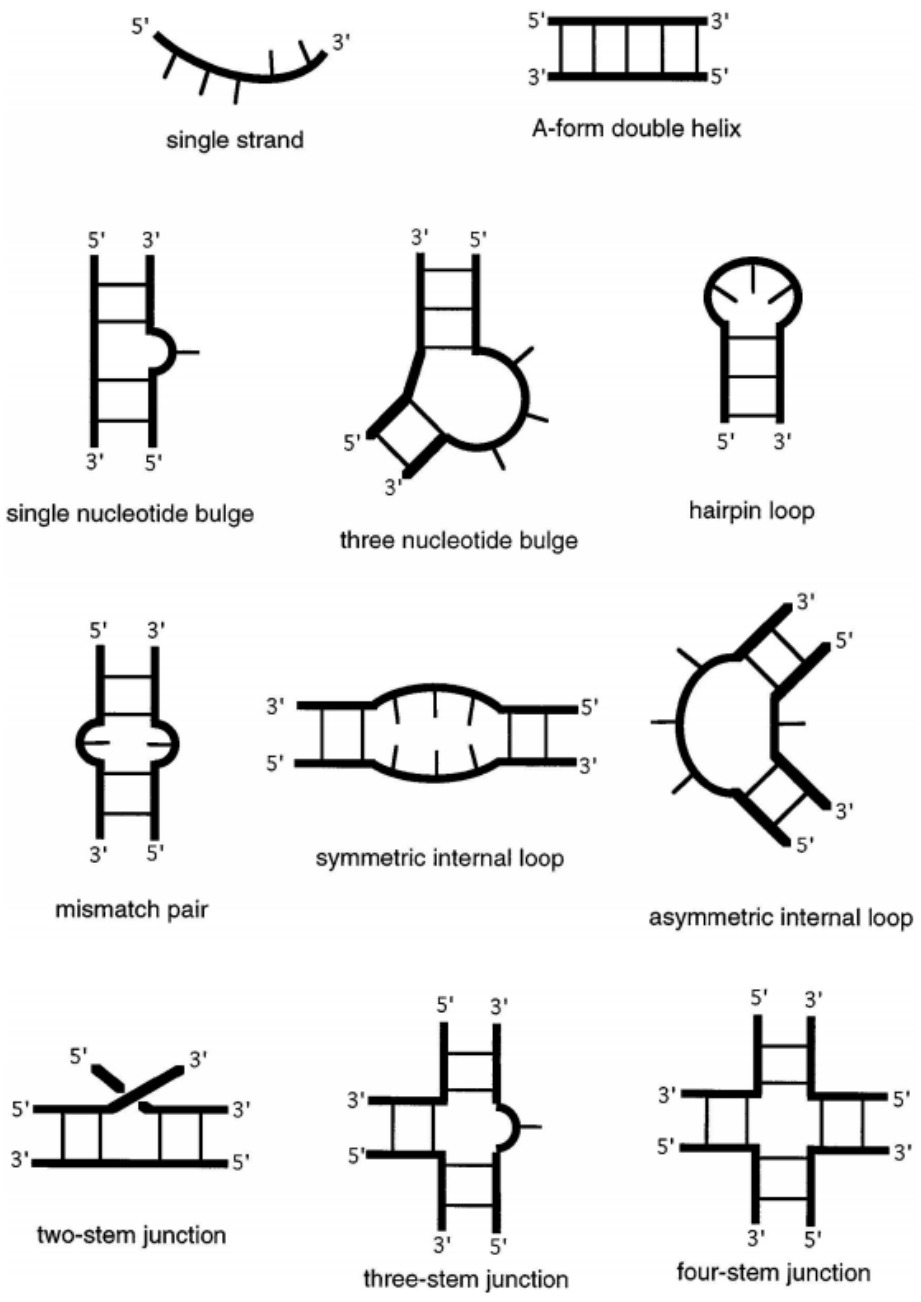


Figure 5: Secondary Structure Motifs *(Hoksza, et al., 2011)*

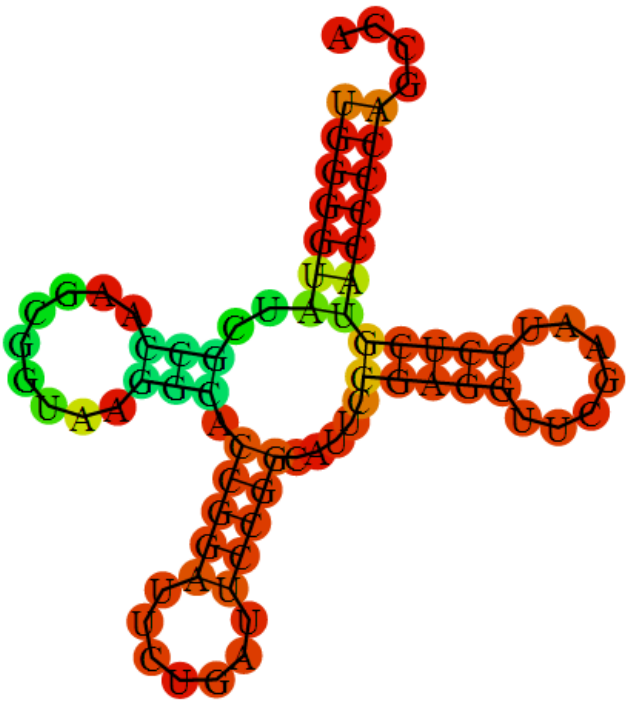


Figure 6: The secondary structure of 1QRS; *generated by http://rna.tbi.univie.ac.at/cgi-bin/RNAfold.cgi*

2.2.2.3 Tertiary structure

Under appropriate conditions, structured RNA molecules undergo a transition to a 3D fold in which the helices and the unpaired regions are precisely organized in space. This tertiary interactions stabilize the overall arrangement and packing of double helices in large RNA structures. The enumeration of each atoms including their three dimensional coordinates in an RNA structure is a description of the tertiary structure. This level of organization is relevant for biological function of structured RNA molecules.

There are three important methods to determine the tertiary structure; the first method relies on X-ray crystallography of single crystals of purified RNA molecules; the second method is nuclear magnetic resonance (NMR); finally, the phylogenetic method coupled to computer modeling and experimental approaches is described. (Westhof, et al., 2000), (Efficient RNA pairwise structure comparison by SETTER method., 2011)

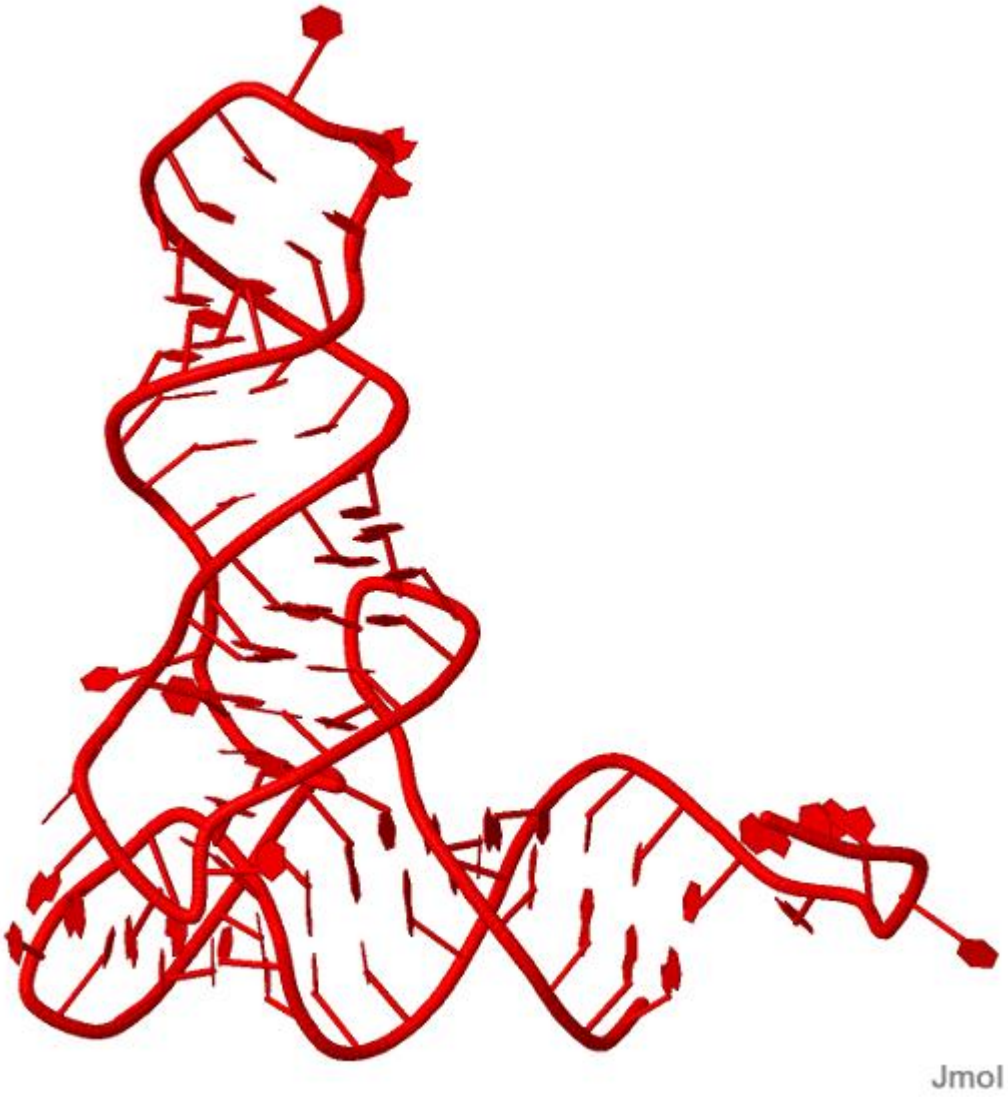


Figure 7: The tertiary structure of 1QSR

3 RNA Alignment

3.1 Why to Alignment RNA Structures

In bioinformatics, an alignment is a way to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. As the RNA structure can be considered as a long sequence of information - in primary structure it is a sequence of base nucleotides, in tertiary a sequence of atoms including coordinates – it is hard to process by manpower, but easy to process by computers. Although, for final decisions the human intelligence, experience and intuition are indispensable, computer science assists to biologists.

3.2 Alignment Methods

3.2.1 Sequential Alignment

Sequential alignment is the most basic method to identify regions of similarity that may be consequence of functional, structural or evolutionary relationship between nucleic acids. As the name implies, it uses the primary structure of a nucleic acid, i.e. the linear sequence of nucleotides.

How can we align two sequences of nucleotides? Just like two sequences of regular characters. In 1966 Vladimir Levenshtein introduced a method for pairwise string alignment. The Levenshtein distance is a string metric for measuring the distance between two sequences. It is defined as minimum number of edit operations to transform one string into the other, with allowable edit operations being insertion, deletion and substitution of a single character. In fact, Levenshtein just introduced the definition of edit distance but never described an algorithm for finding the edit distance between two strings. This algorithm has been discovered and rediscovered many times in applications ranging from automated speech recognition to molecular biology. Although the details of the algorithms slightly vary across the various applications, they are all based on dynamic programming.

The alignment of the strings *v* (of *n* characters) and *w* (of *m* characters) is a two-row matrix such that the first row contains the characters of *v* in order while the second row contains the characters of *w* in order, where spaces may be interspersed throughout the both strings in different places. As a result, the characters in each string appear in order, though not necessarily adjacently. We also assume that no column of the alignment matrix contains spaces in both rows, so that the alignment may have at most $n + m$ columns.

A	T	-	G	T	T	A	T	-
A	T	C	G	T	-	A	-	C

Figure 8: The alignment of ATGTTAT and ATCGTAC sequences

Cells that contain the same letter in both rows are called matches, while columns containing different letters are called mismatches. The columns of the alignment containing one space are called indels.

Analyzing the quality of an alignment is equivalent to analyzing the quality of the corresponding path in the edit graph computed with dynamic programming. Given any two strings, there are a large number of different alignment matrices and corresponding paths in the edit graph. Some of these have a surplus of mismatches and indels and a small number of matches, while others have many matches and few indels and mismatches.

(Jones, et al., 2004 pp. 166-185)

3.2.2 Multiple Sequence Alignment

Functionally similar nucleic acids may not exhibit a strong sequence similarity (although they can show a high level of structural similarity). If sequence similarity is weak, pairwise alignment can fail to identify functionally related sequences because weak pairwise similarities may fail statistical tests for significance. However, simultaneous comparison of many sequences often allows one to find similarities that are invisible in pairwise sequence comparison.

Just like pairwise sequence alignment, multiple sequence alignment can be solved by dynamic programming. To solve the problem a multiple alignment matrix was introduced, which is a generalization of the pairwise alignment matrix. The main disadvantage of this method is the computational complexity.

(Jones, et al., 2004 pp. 185-193)

For more efficient solution of multiple sequence alignment, bioinformatics had to introduce some new approaches, for example ClustalW.

3.2.3 ClustalW

The base idea of ClustalW consists of three main stages:

1. All pairs of sequences are aligned separately in order to calculate a distance matrix giving the similarity of each pair of sequences
2. A guide tree is calculated from the distance matrix
3. The sequences are progressively aligned according to the branching order in the guide tree

In the following sections we will describe the stages in a bit more detail.

The distance matrix is relatively simple. The only task is to calculate the distance between all pairs of sequences using a pairwise sequence alignment algorithm. The resulting distances are then represented by the all-by-all distance matrix.

The tree used to guide to the final alignment process is calculated from the distance matrix using the Neighbor-Joining method.

There could be a question, why to use a guide tree. The reason is that the most divergent sequences (most different on average from all of the other sequences) are usually the most difficult to align correctly. It is sometimes better to delay the incorporation of these sequences until all of the more easily aligned sequences are merged first.

The progressive alignment is done by using series of pairwise alignment to align groups of sequences according the guide tree. Each step consists of aligning two existing alignments or sequences.

The output of a ClustalW algorithm is a sequence of nucleic bases, an alignment that contains significant regions of the input structures.

(CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, osition-specific gap penalties and weight matrix choice, 1994)

3.3 Structural Alignment Methods

Since the function of RNA is more correlated to the three dimensional layout, then the order of nucleotide bases, to properly understand the function of RNA molecules requires methods for comparing and analyzing their 3D structures. Although detecting optimal structural similarity between two biomolecules in 3D has been shown to be NP-hard, the development of automatic tools capable of an efficient and accurate RNA structural alignment has become an important part of structural bioinformatics.

Currently available software tools for comparing two RNA 3D structures, such as ARTS, DIAL, iPARTS, SARA, SARSA or R3D Align are based on heuristic approaches. ARTS detects a maximum common substructure between two RNA 3D structures using backbone phosphate atoms. ARTS is not practical for comparison of large RNA molecules due to its cubic time complexity. The DIAL server using a dynamic programming algorithm and running in a quadratic time. The algorithm is based on torsion and/or pseudotorsion similarity sequence similarity and base pairing similarity. An improvement in the speed over the DIAL algorithm was later brought by PARTS, an algorithm based on the so-called structural alphabet (SA). SA is an emerging concept in the structural biology of proteins. A protein structure is represented as a limited series of ‘letters’ each assigned to a well-characterized conformation. The RNA structures are represented as 1D sequences of SA letters, and these are aligned using classical methods for pairwise and multiple sequence alignments. A new set of SA letters was derived for the improved version of PARTS called iPARTS. In SARA, distances among selected atoms are represented as unit vectors existing on unit spheres. All-to-all unit vector Root Mean Square Division (RMSD) distances of consecutive unit spheres are computed and used as scoring matrix for the dynamic programming-based global alignment. The R3DAlign is

based on local nucleotide by nucleotide superpositions that effectively accommodate the flexibility of RNA molecules. Local alignments are then merged to form a global alignment.

(Efficient RNA pairwise structure comparison by SETTER method., 2011 pp. 1-2)

3.3.1 SETTER

SETTER (SEcondary sTructure-based TERtiary Structure Similarity Algorithm) is a novel pairwise RNA comparison method is introduced by David Hoksza and Daniel Svozil. This method divides the whole RNA structure into a set of non-overlapping generalized secondary structure units (GSSUs). The distance measure based on RMSD transformations between all possible pairs of GSSUs is utilized to get the resulting RNA structure comparison. The comparison algorithm scales as $O(n^2)$ with the size of GSSU, and as $O(n)$ with the number of GSSUs in the structure. The segmentation to GSSUs offers the advantage of an exemplary runtimes even for the largest structures.

Considering the impressive time complexity and precision compared with the above-described methods, the SETTER algorithm has been chosen as the base alignment method for this new multiple structural alignment.

In order to understand the idea behind the MultiSETTER algorithm, the SETTER has to be described in details from (Efficient RNA pairwise structure comparison by SETTER method., 2011) – copied.

3.3.1.1 GSSU Identification

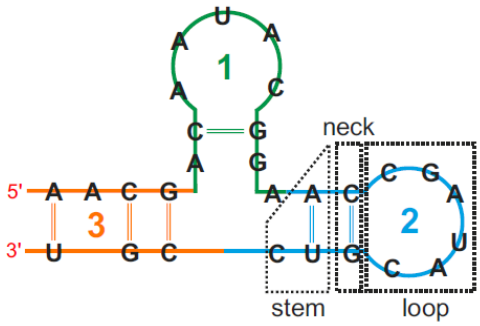


Figure 9: Three GSSUs extracted from an RNA structure. The sequence starts at the 5' end and the numbers denote order of GSSU generation.

GSSU usually resembles a hairpin motif possibly containing bulges and/or internal loops in its stem part. Three important elements are recognized in the GSSU: loop, neck, and stem. The formal description of the GSSU is given by the following definition.

Let $\mathcal{R}$ be an RNA structure with nucleotide sequence $\{n_i\}_{i=1}^{n}$ and let $\mathcal{WC} \subset \mathcal{R}$ denotes the subset of n_i participating in a Watson-Crick base pair. By generalized secondary structure unit (GSSU) $\mathcal{G}$, we understand a pair of substrings of $\mathcal{R}$, $\{n_i\}_{i=i_1}^{i_2}$ and $\{n_j\}_{j=j_1}^{j_2}$ $(i_1 \leq i_2 < j_1 \leq j_2, i_2 = j_1 - 1)$ of maximum lengths such that each nucleotide $n_x \in \mathcal{G}$:

- $i_1 \leq x \leq i_2$: $n_x \notin \mathcal{WC}$ or n_x is paired with n_y where $j_1 \leq y \leq j_2$
- $j_1 \leq x \leq j_2$: $n_x \notin \mathcal{WC}$ or n_x is paired with n_y where $i_1 \leq y \leq i_2$

Let i_{max} and j_{min}be the highest/lowest indexes of the Watson-Crick paired bases in $\mathcal{G}$. We define **loop** as $\mathcal{L} = \{n_i\}_{i=i_{max}+1}^{j_{min}-1} \subset \mathcal{G}$, **stem** as $\mathcal{G} \backslash \mathcal{L}$ and neck as the pair $\{n_{i_{max}}, n_{j_{min}}\}$.

For the purpose of the algorithm, each nucleotide in GSSU is represented by its C4' atom, and Watson-Crick (WC) hydrogen bonds are identified using 3DNA. Note that even a structure without a single Watson-Crick pair has a GSSU that is identical with the structure itself.

3.3.1.2 Comparing Two GSSUs

GSSU pairwise comparison lies in the heart of the method. It is employed even when SETTER compares structures consisting of multiple GSSUs.

Each GSSU is represented by an ordered set of 3D coordinates enhanced with bonding and nucleotide/atom type information. The common way how to assess similarity between two sets of points is to define a pairing between them. The sets are superposed by finding translation and rotation of one structure over the other by minimizing the mutual distance of the respective paired points. Usually, the root mean square deviation (RMSD) is used as the distance measure, and two structures can be superposed given a pairing/alignment in polynomial time. However, finding the optimal alignment (which in turns minimizes the distance measure) is an NP-hard problem. The optimal solution can be found by exhaustive search, which is, however, computationally not feasible. This problem can be resolved by identifying suboptimal alignments that will likely participate in the optimal

alignment. This is the principle idea behind SETTER's structure comparison process. Working with relatively short alignments allows to superpose even the largest RNA structures in a reasonable amount of time.

To superpose two GSSUs, is to match their loops, which implies matching their necks. To define the superposition unambiguously in three-dimensional space at least three pairs of points are necessary. A set of these points is called triplet, and the alignment is formed by matching triplets between two given structures. SETTER first aligns necks and then tries to identify the final pair in the triplet by aligning each possible pair of loops' nucleotides. For example, if two GSSUs with loops consisting of n and m nucleotides are to be aligned, $n \times m$ alignments are generated (See Figure 10)

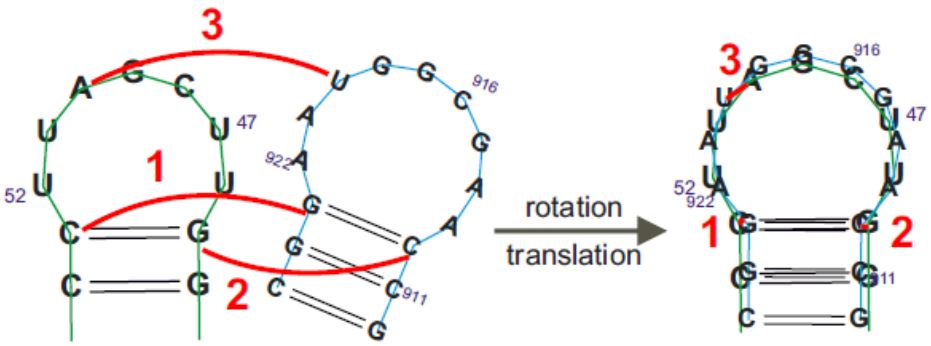


Figure 10: Alignment of GSSU from tRNA domain of transfer-messenger RNA (PDB code 1P6V) with GSSU from glutamine tRNA (PDB code 1EXD). The final structural alignment is defined by three nucleotide pairs forming a triplet (the lines 1, 2, and 3). To find the optimal superposition for the given neck pairs (lines 1 and 2), the position of the middle pair is varied (line 3).

For each alignment, a rotation matrix and a translation vector defining optimal superposition of the triplets is calculated, and though these are optimal for given triplet only, they are subsequently used to superpose the whole GSSUs. This possible inaccuracy is a trade-off for efficiency. After the superposition, a 3D nearest neighbors are found for each nucleotide in the aligned GSSUs, and their distances are added to the overall distance of the two GSSUs. Finally, the distance is normalized resulting in a measure referred to as $\mathfrak{S} - distance$.

Though in most cases the GSSU consists of a stem and a loop, it is not a strict rule, as demonstrated by GSSU number 3 in Figure 9. Two particular situations can occur — GSSU has a zero-sized loop or the RNA does not have a single hydrogen bond (i.e. it does not have a secondary structure at all). In case of GSSU without the loop the third nucleotide for triplet alignment is selected from the stem and its position within the stem is varied to get the lowest $\mathfrak{S} - distance$ of the superposed GSSUs.

3.3.1.3 Comparing More Than Two GSSUs

In case when one of the structures contains more than one GSSU the following three-step multiple GSSU comparison process is implemented:

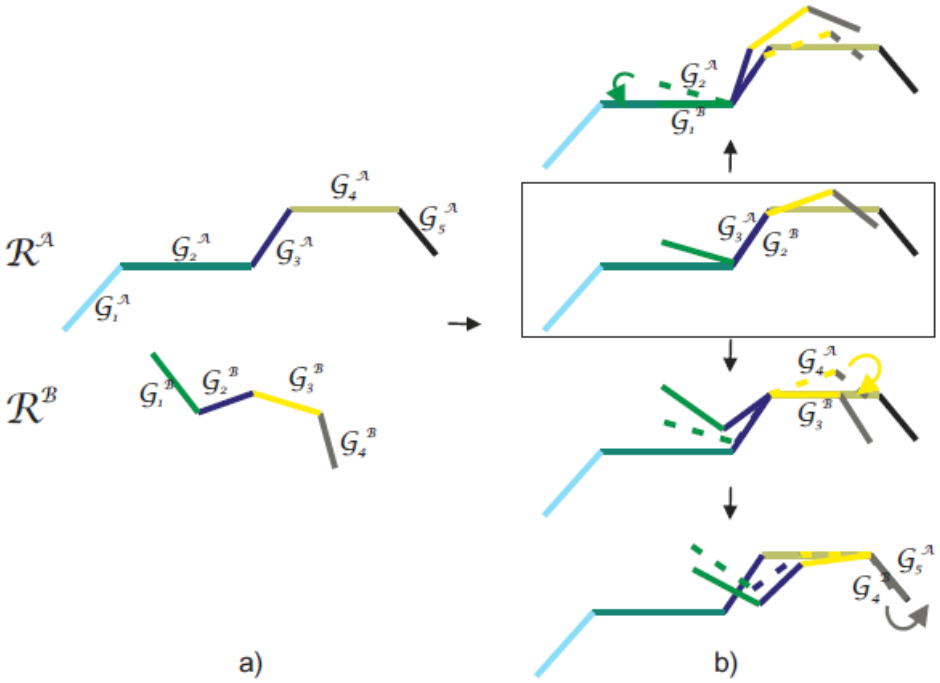


Figure 11: Multiple GSSU structure comparison works in four steps. The figure represents a situation where only the optimal GSSU pair is considered ($\mathcal{K} = 1$), and the direction of the alignment is given by the order of GSSUs. A schematic representation of two RNA structures $\mathcal{R}^{\mathcal{A}}$ and $\mathcal{R}^{\mathcal{B}}$ with 5 and 4 GSSUs, respectively. The most similar GSSUs pair is $\{\mathcal{G}_3^{\mathcal{A}}, \mathcal{G}_2^{\mathcal{B}}\}$ (shaded rectangle). Structures $\mathcal{R}^{\mathcal{A}}$and $\mathcal{R}^{\mathcal{B}}$are aligned based on the rotation and translation of this pair. The superposition needed to optimally align $\mathcal{G}_4^{\mathcal{A}}$ and $\mathcal{G}_3^{\mathcal{B}}$ was obtained during the all-to-all GSSU pairwise comparison stage, and the rotation angle needed to get from the state $\{\mathcal{G}_3^{\mathcal{A}}, \mathcal{G}_2^{\mathcal{B}}\}$ to $\{\mathcal{G}_4^{\mathcal{A}}, \mathcal{G}_3^{\mathcal{B}}\}$ (first down arrow in the figure) is thus

known. The current state is changed to $\{\mathcal{G}_4^{\mathcal{A}},\ \mathcal{G}_3^{\mathcal{B}}\}$, and the process is repeated for the pair $\{\mathcal{G}_5^{\mathcal{A}},\ \mathcal{G}_4^{\mathcal{B}}\}$ (second down arrow in the figure). Similarly, the algorithm must also process in the opposite direction from the position of $\mathcal{G}_2^{\mathcal{A}}$ and $\mathcal{G}_1^{\mathcal{B}}$ (up arrow in the figure).

1. Perform all-to-all pairwise GSSU comparisons.
2. Few best GSSU pairs (with low $\mathfrak{S} - distance$) are used as seeds for alignment of the rest of the GSSUs.
3. The structures' GSSUs are aligned (each treated as a whole) and the multiple GSSU alignment with the lowest aggregated $\mathfrak{S} - distance$ is declared as the overall pairwise structure similarity score.

Let us compare structures $\mathcal{R}^{\mathcal{A}}$ and $\mathcal{R}^{\mathcal{B}}$ having the distance between them denoted as $\overline{\mathfrak{S}}$. Each GSSU from $\mathcal{R}^{\mathcal{A}}$ is compared to each GSSU from $\mathcal{R}^{\mathcal{B}}$, but only top $\mathcal{K}$ pairs with the minimum distance is processed further. For each of the $\mathcal{K}$ selected pairs $\{\mathcal{G}_i^{\mathcal{A}},\ \mathcal{G}_j^{\mathcal{B}}\}$ the value of $\mathfrak{S}$ is set to $\mathfrak{S}\{\mathcal{G}_i^{\mathcal{A}},\ \mathcal{G}_j^{\mathcal{B}}\}$. In the second step, $\mathfrak{S} - distance$s of the neighboring GSSU pairs are iteratively added to the $\overline{\mathfrak{S}}$. For the GSSU pair $\{\mathcal{G}_{i+1}^{\mathcal{A}},\ \mathcal{G}_{j+1}^{\mathcal{B}}\}$ the value of $\mathfrak{S}\{\mathcal{G}_{i+1}^{\mathcal{A}},\ \mathcal{G}_{j+1}^{\mathcal{B}}\}$ and the penalty for the rotation needed to transform the structures from the state corresponding to $\{\mathcal{G}_i^{\mathcal{A}},\ \mathcal{G}_j^{\mathcal{B}}\}$ to the state corresponding to $\mathfrak{S}\{\mathcal{G}_{i+1}^{\mathcal{A}},\ \mathcal{G}_{j+1}^{\mathcal{B}}\}$ are added to $\overline{\mathfrak{S}}$. This process goes from $\{i+1, j+1\}$ until either $i+1$ or $j+1$ reaches the number of GSSUs in $\mathcal{R}^{\mathcal{A}}$ or $\mathcal{R}^{\mathcal{B}}$. Similarly, the other ends of the structures need to be aligned and so the process is repeated for $\{i-1, j-1\}$. However, the case when GSSUs in the RNA structure are oriented in opposite direction must also be considered, and another $\mathcal{K}$ alignments must be performed aligning $i-1$ residues with $j+1$ residues (not shown in Figure 11).

The rotation between two GSSUs imposes a penalty to the $\overline{\mathfrak{S}}$. This penalty is calculated as a distance between the rotation matrices defining the two consequent GSSU superpositions. The penalty for translation is, on the other hand, not included explicitly, as it is already present in the pairwise GSSU $\mathfrak{S} - distance$ where the difference in GSSUs' sizes is used for normalization (see section 3.3.1.2). This approach to multiple

GSSU comparison allows matching any pair of structures including the largest ones in a reasonable amount of time.

In order to increase the algorithm's speed a simple early termination condition has been implemented into the SETTER's GSSU comparison process.

4 Multiple Structural Alignment

4.1 Motivation

Understanding the architecture and function of RNA molecules requires methods for comparing and analyzing their structures. Since the function of an RNA molecule is largely determined by its 3D structure, they are typically more evolutionary preserved than sequence, methods for comparative RNA function annotation based on the structural similarity usually yield much better results than sequence based approaches. Although a structural alignment of short RNAs is achievable in a reasonable amount of time, large structures represent much bigger challenge. However, pairwise alignment can fail on functionally related but evolutionary distant RNA structures. To determine the real functionality, it is necessary to treat a family, as a whole unit.

Let's consider a real world situation: a biologist discovers a new RNA structure. This means, that the three dimensional positions of atoms are known, however the function of RNA is unknown. What can the biologist do? He can compare the three dimensional structures of the newly found RNA to families with known function.

Therefore, the purpose of this algorithm is to align a family of functionally correlated RNA structures with a RNA of unknown function to determine whether the RNA belongs to the functional family, respectively shares the same functionality. The RNA with unknown function will be denoted as **test RNA**.

How can this be done? There are several ways to solve this problem:

i.) Compute the distance of test RNA with each RNAs from the given family; process the results by statistical methods to determine whether the test RNA has the same function, as the input family has. In other words, we have to determine whether the test RNA belongs to the family or not.

ii.) Work with the family as a single unit and compute the distance between test RNA and the family.

iii.) Align RNA structures from the family including the test RNA and inspect the result.

Distance Calculation and Statistical Approach

This solution seems a little poor, since it uses only pairwise alignments and doesn't operate with the fact, that the input RNAs shares the same function and probably the same structure.

Distance Between Aligned Family And RNA

While applying this solution we have to take into consideration the family of RNA structures as a single unit. To achieve this, we should use the principles used in the ClustalW algorithm:

1. Calculate a distance matrix providing the divergence of each pair of RNA structures.
2. Calculate a guide tree from the distance matrix.
3. Align pairs of RNA structures according to branching order in the guide tree. In this case the output of the alignment is a so-called **average RNA** structure that carries the structural characteristics of both parent RNAs.

The only difference in the current and ClustalW method is, that we need to use structural alignment instead of sequential.

Align Family Including RNA

This can also be done using the previously described algorithm. The result could be calculated from distances between average RNA structure and RNAs from the input family using some statistical methods. The other possible solution is to calculate the average RNA, then superpose input RNA structures including the test RNA; visualize the results and let biologists make the final decision.

The main disadvantage of this method is the fact that newly discovered RNA structure can affect the computation in bad manner.

Conclusion

The most rational solution for the problem seems the second one. The implementation can assist us while testing the third option as well.

4.2 The Algorithm

In this section we will describe the algorithm that aligns a set of RNA structures and creates an average RNA.

The algorithm takes as an input a set of RNA structures. At first the distance matrix is calculated from the distances between all pairs of input RNA, then a guide tree is constructed using the distance matrix. Finally, according the guide tree the algorithm merges input RNA structures into an average RNA, which is the result of the alignment.

4.2.1 Distance Matrix Calculation

The distances between each pair of RNA are computed using the SETTER algorithm. In order to retrieve data consumed by subsequent parts of the algorithm, the SETTER algorithm was modified, however the functionality remained the same.

4.2.2 The Guide Tree

The importance of theThe tree has been already debated when we discussed the ClustalW algorithm. In this section the Neighbor Joining algorithm will be presented in details.

4.2.2.1 Neighbor-Joining

The Neighbor-joining is proposed for reconstructing phylogenetic trees from evolutionary distance data. In the construction of phylogenetic trees, the principle of minimum evolution or maximum parsimony is often used. The standard algorithm of the tree-making methods based on this principle is to examine all possible topologies (branching patterns) or a certain number of topologies that are likely to be close to the true tree and to choose one that shows the smallest amount of total evolutionary change as the final tree.

The principle of this method is to find pairs of operational taxonomic units (OTUs = neighbors) - in our case RNA structures - that minimize the total branch length at each stage of clustering of OTUs starting with a starlike tree. It is possible to define the topology of a tree by successively joining pairs of neighbors and producing new pairs of neighbors.

Algorithm takes as input a distance matrix specifying the distance between each pair of OTUs. The result is an unrooted tree.

4.2.2.1.1 Notations

- Let D be the distance matrix, and d_{ij} be the distance between the i^{th} and j^{th} taxon.
- Let r the number of taxa
- Let Q be a helper matrix computed from D to compensate for long edges by subtracting the averaged distances to all other leaves.

$$q_{ij} = d_{ij} - \frac{1}{r-2}\left(\sum_{k=1}^{r} d_{ik} + \sum_{k=1}^{r} d_{jk}\right)$$

4.2.2.1.2 The Algorithm

1. Based on the current distance matrix calculate the Q matrix
2. Find a pair of taxa (i, j) with the lowest q_{ij} value. Add a new node to the tree by joining the selected taxa.
3. Calculate the distance from recently paired taxa to this new node

$$d_{fu} = \frac{1}{2} d_{fg} + \frac{1}{2(r-2)} \left[\sum_{k=1}^{r} d_{fk} - \sum_{k=1}^{r} d_{gk}\right] \text{ and}$$

$$d_{gu} = d_{fg} - d_{fu}$$

where f and g are the recently paired taxa and u is the newly generated node.

4. Calculate the distance from each of the not recently paired taxa to the new node

$$d_{uk} = \frac{1}{2}\left[d_{fk} + d_{gk} - d_{fk}\right]$$

where u is the new node, k is the node for which we want to calculate the distance and f and g are the members of the pair just joined.

5. Replace the paired taxa in D with the new node and decrease the value of r.
6. Repeat steps 1-5 until $r = 2$

The result of the neighbor joining method is an unrooted tree.

(The Neighbor-joining Method: A New Method for Reconstructing Phylogenetic Trees, 1987)

4.2.2.2 Modifications

Bearing in mind, that the MultiSETTER algorithm creates a new average RNA from each node paired by neighbor-joining method, it could be beneficial to recalculate the guide tree after a new node - average RNA – was added to the tree.

The neighbor-joining method is applied after each iteration of merging algorithm for the input set of RNA structure excluding the recently paired RNAs and including the newly created average RNA. This modification could help to reorganize the guide tree based on the most recent state of the alignment to get better results.

4.2.3 Merging Two RNAs

Since this multiple alignment algorithm is based on aligning – in other words merging – two RNAs, therefore this is the most important stage of the MultiSETTER algorithm. It is crucial to keep the structural characteristics of both input RNAs while merging. In order to understand the concept of merging it is essential to understand the SETTER algorithm (3.3.1).

To correctly merge two RNAs it is important to know which GSSUs to merge. The source of this information is the SETTER algorithm. The GSSU pairs that are being merged have to contribute in the lowest aggregated $\mathfrak{S} - distance$. Other important data retrieved from SETTER are the results of the alignment like distance, translation vector and rotation matrix belonging to GSSU pairs.

To reduce the negative impact of outlier RNAs the algorithm distinguishes the two input RNAs. The one, which is closer to other RNAs from the input set, is the **pattern RNA**, to which the other RNA is aligned (superposed). Thus, the pattern RNA is the guide of the merging process.

The next step is to convert GSSUs of the pattern RNA to **sub RNA** structures that contain only the residues of the actually processed GSSU. If the GSSU is contributed in the lowest aggregated $\mathfrak{S} - distance$, the GSSU, respectively the sub RNA, is merged with its GSSU pair from the other RNA.

Finally, merged and not merged sub RNAs are appended in the same order, as they were in the pattern RNA.

Notations:

- **qRNA** – the pattern RNA
- **dbRNA** – the RNA that is aligned to the pattern RNA
- **q** prefix – variable derived from qRNA
- **db** prefix – variable derived from dbRNA

Pseudo Code:

```
 1. for each GSSU in qRNA
 2.   qSubRNA := convertToRna(GSSU);
 3.   rSubRNA := qSubRNA;
 4.   if GSSU is in GSSUPairs then
 5.     dbSubRNA := convertToRna(GSSUPairs[GSSU].other);
 6.     if dist(qSubRNA, dbSubRNA) < meanDistance * param1 then
 7.       rSubRNA := merge(qSubRNA, dbSubRNA);
 8.       if dist(rSubRNA, qSubRNA) + dist(rSubRNA, dbSubRNA) <
 9.         dist(qSubRNA, dbSubRNA) * param2 then
10.         rSubRNA := qSubRNA;
11.       end if;
12.     end if;
13.   end if;
14.   averRNA.append(rSubRNA);
15. end for;
```

4.2.3.1.1 Quality Improvements

To improve the quality of the alignment, two criteria has to be applied to reduce the negative impact of outlier RNAs:

- Merging is avoided if the distance between parent GSSU pairs is greater than the mean of distances between GSSU pairs of each RNA from the input set, contributed in the lowest aggregated $\mathfrak{S} - distance$.
- Merged GSSU are rejected if the distance between the result and parent GSSUs is smaller than then the distance between parents GSSUs multiplied by a parameter.

In these cases the appended **sub RNA** is the one, which is converted from the pattern GSSU.

4.2.3.2 Merging Two GSSUs

As already stated, the GSSU contains three important elements: loop, neck and stem. Actually the neck could be considered as a part of the stem. The algorithm merges loops and stems separately.

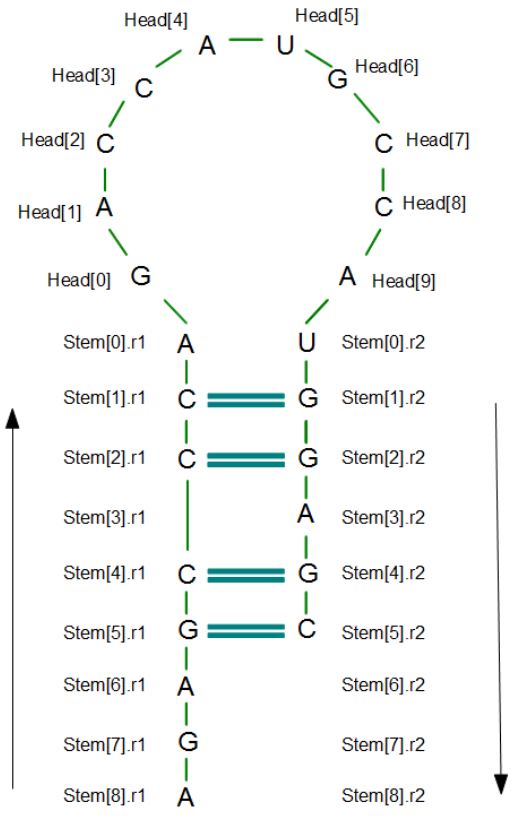


Figure 12: Indexing of GSSU

According the definition of GSSU, the sequential order of residues in RNA is the fallowing: stem[n - 1].r1, stem[n - 2].r1, …, stem[1].r1, stem[0].r1, head[0].r1, head[1].r1, …, head[m - 2].r1, head[m - 1].r1, stem[0].r2, stem[1].r2, …, stem[n - 2].r2, stem[n - 1].r2, where n is the length of stem and m is the length of head. Since the representation of RNA in SETTER is based on the sequential order of residues, GSSUs must be processed in the same order.

4.2.3.2.1 Preprocessing

To acquire the triplets of two input RNAs to the same position, the dbRNA has to be rotated by rotation matrix and translated by translation vector.

4.2.3.2.2 Merging the "r1 Side" Stems

```
 1. Let Stem1 be the longer stem from qGSSU.Stem and dbGSSU.Stem;
 2. Let Stem2 be the shorter stem from qGSSU.Stem and dbGSSU.Stem;
 3.
 4. length := Stem1.length;
 5. vector 3d translationVector = (0, 0, 0);
 6. if Stem2.length = 0 then
 7.   resultGSSU.r1Side := Stem1.r1Side;
 8.   return;
 9. else
10.   double step := Stem1.length / Stem2.length;
11.   for i := length – 1 downto 0 do
12.     residue1 := Stem1[i].r1;
13.     residue2 := Stem2[step * i].r1;
14.     // selects residue according the merging rules
15.     pos := i;
16.     resultResidue := SelectResidue(residue1, residue2, pos);
17.     if (residue1 != null and residue2 != null) then
18.       translationVector :=
19.         CalcTranslation(residue1,residue2, resultResidue);
20.       trasnalationSourceRNA := resultResidue.ownerRNA;
20.     else
21.       if resultResidue.ownerRNA = trasnalationSourceRNA then
22.         tranlationDir := 1;
23.       else
24.         tranlationDir := -1;
25.       end if;
26.     end if;
27.     TransleteAtoms(resultResidue.atoms,
28.       tranlationDir * translationVector);
29.     resultGSSU.atoms.append(resultResidue.atoms);
30.     resultGSSU.Stem[i].r1 := resultResidue;
31.   end for;
32. end if;
```

The algorithm merges the r1 side of two stems. The initial length of the new stem is the maximum of lengths of the two input stems.

4.2.3.2.3 Residue Selection

To select the actually added residue (line 16.) two algorithms were implemented:

- Favors residues from the pattern GSSU – meaning, that residue is selected from dbGSSU only if there is no residue in qGSSU.

- Both GSSU has the same privileges – the decision is dependent on the position (order number) of the merged residue. For this purpose modulo by two is used.

The following diagram contains the merging rules:

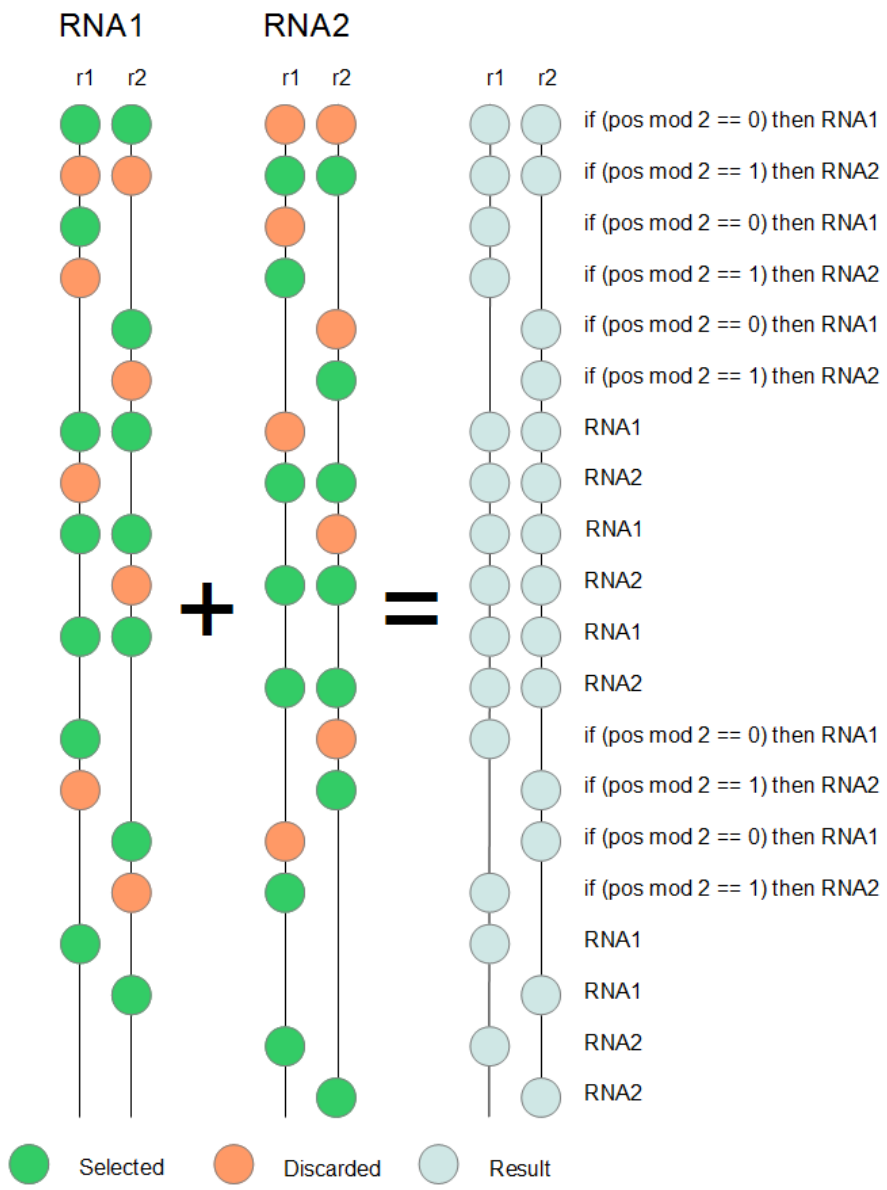


Figure 13: GSSU Stem merging rules

It is crucial to position the atoms of the new residue correctly. In order to achieve this a translation vector is calculated. If both residues exist, the vector translates the selected residue to the midpoint between residues. The direction of the translation vector - in our

case is the RNA containing the selected residue - is also stored. This data is used when only the selected RNA has residues. The translation vector is not modified if the existing residue is a part of the same RNA as the translation vector's source RNA; otherwise the translation vector is multiplied by *-1* to redirect it (line 21).

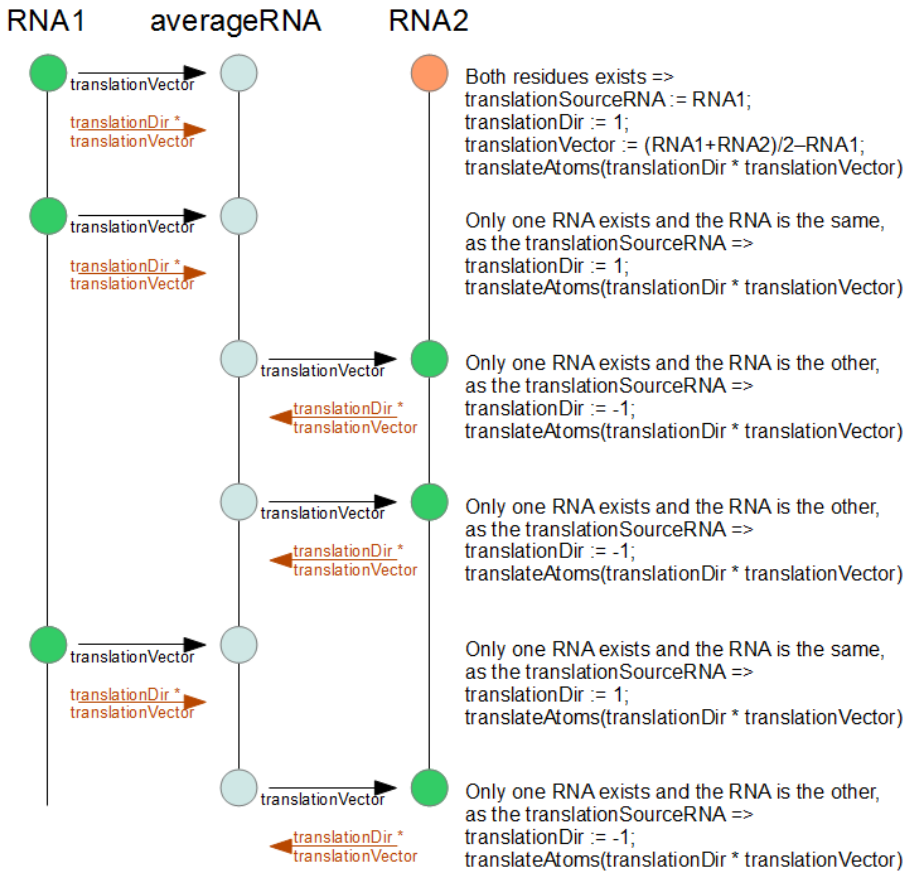


Figure 14: Residue translation

4.2.3.2.4 Merging two Heads

```
 1. Let Head1 be the longer stem from qGSSU.Head and dbGSSU.Head;
 2. Let Head2 be the shorter stem from qGSSU.Head and dbGSSU.Head;
 3.
 4. length := Head1.length;
 5. if Head2.lenght = 0 then
 6.   resultGSSU.Head = Head1;
 7. else
 8.   double step := Head1.length / Head2.length;
 9.   for i := 0 to length – 1 do
10.     residue1 := Head1[i];
11.     residue2 := Head2[step * i];
12.     if i mod 2 = 0 then
13.       resultGSSU.Head[i] := residue1;
14.     else
15.       resultGSSU.Head[i] := residue2;
16.     end if;
17.     resultGSSU.Head[i].coord :=
18.       (residue1.coord + residue2.coord) / 2;
19.     resultGSSU.atoms.append(resultGSSU.Head[i].r1.atoms);
20.   end for;
21. end if;
```

The algorithm is approximately the same as the above described. The only difference is that the head of GSSU cannot contain mismatches; therefore the residues can't be *null*.

4.2.3.2.5 Merging the "r2 side" of Stems

The algorithm differs from the algorithm used to merge r1 Side in three respects:

- The direction of for loop (line 11.)
- r1 is swapped by r2
- If the residues of currently processed base pairs are *null* on both sides, the base pair is removed from the result GSSU (line 15.)

5 Technical Details

This chapter describes some interesting technical details of the implementation.

5.1 Two Tier Model

Since the base algorithm, the SETTER, has been implemented in C++ there were no questions what programming language to choose to implement the MultiSETTER algorithm. The benefits of using C++ are the control over memory consumption, more effective code, and possibility of parallelization. The main disadvantage of C++ is that we live in a "Managed World". There are dozens of nice Java applications working with nucleotide acids. Java is easy to develop, easy to integrate existing codes, easy to deploy to the Internet.

The above-mentioned facts were leading to the decision to split the application to two individual parts; where the engine has been written in C++ and a GUI in Java.

5.2 Engine

The Engine, respectively its algorithm, is the main objective of this work. This fragment is responsible for all the computation.

5.2.1 Parallelization

To reduce the calculation time of MultiSETTER it was essential to take advantage of the multicore processors. Intel's Thread Building Blocks were chosen for this purpose. It is easy to use and effective header for parallelization.

5.2.1.1 SETTER

As the algorithm is using SETTER to provide pairwise alignment, the first step was trying to parallelize it. At the first look SETTER seems easy to parallelize, the algorithm contains functions with embedded for cycles, which calls other functions with the same structure. On the other hand, an early termination criteria has been implemented to the SETTER's GSSU comparison process in order to reduce the calculation time. Each embedded function takes as an input parameter the result of the previous call. This implies a fast and accurate algorithm that is hard to parallelize.

There are two possible solutions for this problem:

- Omit the early termination condition – the results are accurate, but the algorithm slows down significantly.
- Using a variable that contains the lowest results from previous iterations. This can be implemented using mutexes, which will slow down the algorithm. The other solution would be updating the variable at the end of parallel for chunks. In this case the result will be hardly dependent on the chunk size.

The consequence is that we have to try parallelize the algorithm somewhere else.

5.2.1.2 Reading the Input

The size of the input PDB files (6.1.2.1) could reach several megabytes, therefore reading and parsing them could be very time-consuming. From the parallelization point of view, conflict could occur when two threads simultaneously access the same file, however the MultiSETTER only reads the files, and therefore the issue is resolved. Parallel for is used to parallelize the reading process.

5.2.1.3 All-to-All / All-to-One Pairwise Alignments

MultiSETTER algorithm often aligns multiple RNA structures in the same block of code:

- When the distance matrix is calculated from the input RNA structures.
- When a set of RNA structures are aligned with an average RNA structure.

Each of them are implemented by using parallel for loop. The used data structures are designed in a way that there is no need for mutual executions and thread safe containers.

5.2.1.4 Merging Process

The following facts imply that the merging process is easy to parallelize:

- At the beginning of RNA merging is already determined which GSSU pairs to merge (the ones that are contributed in the lowest aggregated $\mathfrak{S} - distance$).
- Merging two GSSUs is an isolated problem that has no effect for merging other GSSUs.

The parallelization is done by parallel for.

5.2.1.5 Chaining GSSUs into RNA

The merged GSSUs must be appended into the average RNA structure, which seems a good place to use parallel reduce. The parallel reduce splits the problem into subranges, solves them and joins the results of subranges in order to get the final result. In this case the joining algorithm will be very complex, therefore the parallel algorithm will be slower than the sequential.

5.2.2 Storing the Result – Serialization

The role of this application is to help biologist finding the functionality of newly found RNA by aligning them to functional families represented by an average RNA. Thus it would be reasonable to store calculated average RNA structures, since the average computation could be time-consuming. This will allow the users to calculate the average RNA structure once then use it anytime.

The application provides two type of serialization:

5.2.2.1.1 Serialization to PDB File

The serialized PDB file contains the sequence of atoms in the RNA. It is important to mention that the average RNA is a fictive RNA, it is not recommended to use it for other purposes than for visualization.

*5.2.2.1.2 Serialized Internal Objec*t

This type of serialization allows to store all the data generated at merging process. BOOST's Serialization library was chosen for this purpose. This library allows to easily add serialization methods to existing structures/classes. This feature was very important since MultiSETTER uses an existing representation of RNA, which is by SETTER.

5.2.3 The Input

5.2.3.1.1 SETTER's RNA Representation

The SETER application uses the following input files to construct the class which internally represents RNA structures:

- PDB (*.pdb) file (see 6.1.2.1)
- 3DNA (*.out) file (see 6.1.3.1.2)

Since PDB file can contain multiple chains, the constructor takes an extra parameter, the chain ID.

5.2.3.1.2 The Input File's Structure

Engine consumes only well-formed input files. Malformed input can cause an infinite loop or the application can ends with an error. To avoid this, it is recommended to use input files generated by the GUI.

The input file is divided into blocks, each block represents one RNA structure. The block contains the following 5 lines:

1. Absolute path to the PDB file – mandatory
2. Absolute Path to the 3DNA file – mandatory
3. Chain ID – mandatory
4. Comments – not mandatory, could be a blank line
5. New line – mandatory separator of the block

The allowed line endings are Windows format (\r\n) and UNIX format (\n).

5.3 Java GUI

As it was already mentioned, there are a lot of chemical tools written in Java. GUI written in Java provides us the ability to integrate some of those tools. The most notable application is the Jmol (see 6.4), which is an open-source Java 3D viewer for chemical structures. Since the main objective of this work is not the GUI there is no tool integrated.

The main function of this GUI is to properly formalize the input files consumed by the Engine and to present the alignment results for the user.

5.3.1 Directory Structure

The data used by GUI is divided into 4 subdirectories:

- Input – the predefined space to store input files.
- Output – the predefined space to store the results of the alignment.
- RNA – the space where PDB and 3DNA files are stored.
- Temp – for temporary files.

5.3.2 Storing RNAs

The GUI has a list of RNA structures stored at *GUI/data/RNA/fileList.db*. Each row contains the PDB identifier of the RNA, the absolute path to the PDB file and the absolute path to the 3DNA file separated by semicolon. To reduce the data duplicity, the PDB and 3DNA files don't need to be located in *GUI/data/RNA directory*.

5.3.3 Communication Whit Engine

The GUI starts the Engine using external process calls. To display the user the appropriate result, the output of Engine is wrapped. Only lines are displayed, which starts with a highlighter string, which in our case is "JAVA-".

To properly kill the extern process, a thread is dedicated to check whether the "Kill Process" button was clicked. If the button is clicked, the extern process is immediately terminated.

6 User Guide

6.1 Background and Environment Setup

6.1.1 Where to Find RNA Families

A reliable source of functionally correlated RNA families could be the Structural Classification of RNA (SCOR) database.

The Structural Classification of RNA (SCOR) is a database designed to provide a comprehensive perspective and understanding of RNA motif structure, function, tertiary interactions and their relationships. SCOR 2.0.3 provides a survey of the three-dimensional motifs contained in 579 NMR and X-ray RNA structures. This includes 8,270 secondary structural elements which are organized in a directed acyclic graph (DAG) architecture, allowing multiple parent classes for a motif. Users can browse the database or search by PDB or NDB identifier, keyword or sequence. Descriptions and cartoon representations of each of the classes are available. RNA motifs reported in the literature (e.g. Kink turns, S-turns, GNRA loops) are incorporated into the classification.

The database is accessible at http://scor.berkeley.edu.

(Tamura, et al., 2004)

6.1.2 Where to Find RNA Structures

The application is useless without input data. As previously mentioned, the SETTER algorithm constructs internal RNA structure from two input files:

- PDB file
- 3DNA file

6.1.2.1 The Protein Data Bank

The Protein Data Bank (PDB) archive is the single worldwide repository of information about the 3D structures of large biological molecules, including proteins and nucleic acids. The PDB was established in 1971 at Brookhaven National Laboratory and originally contained 7 structures. In 1998, the Research Collaboratory for Structural Bioinformatics (RCSB) became responsible for the management of the PDB. In addition, the RCSB PDB

supports a website where visitors can perform simple and complex queries on the data, analyze, and visualize the results.

The Protein Data Bank is accessible at http://www.pdb.org.

To download multiple RNAs use the http://www.pdb.org/pdb/download/download.do website.

(RSCB)

6.1.2.1.1 File Format

It is a textual file format describing the three dimensional structures of the molecules held in the Protein Data Bank. The PDB format accordingly provides for description and annotation of protein and nucleic acid structures including atomic coordinates, observed sidechain rotamers, secondary structure assignments, as well as atomic connectivity. The PDB file contains one or more chains. Each chain represents a contiguous polypeptide (or polynucleotide), though there may be breaks that are not explicitly recorded due to crystallographic ambiguity. If there is only one chain, it is usually denoted with "A" in the chain ID field; multichain entries are usually labelled "A", "B", etc. Proteases and their peptide-containing inhibitors are often labelled "E" and "I", respectively.

For the SETTER algorithm the most important records of PDB file are ATOM records. Atom records are the lines starting with "ATOM" prefix. They present the three dimensional coordinates of atom, name of the atom, name of residue containing the atom, serial number of the atom in the structure.

6.1.3 3DNA

3DNA is a suite of software programs for the analysis, rebuilding and visualization of three-dimensional nucleic acid structures. At its core, the software uses a simple matrix-based scheme for calculating a complete set of rigid-body parameters that characterize the spatial relationship of the base pairs in DNA and RNA structures.

To download the application, please visit the http://x3dna.org website. To run the application on Windows systems MinGW/MSYS or Cygwin is required.

6.1.3.1.1 Running 3DNA on Windows with Cygwin

1. Install the latest Cygwin with default settings from http://www.cygwin.com.
2. Download 3DNA Release v2.0 for Cygwin from http://forum.x3dna.org/downloads/3dna-download (registration is required)
3. Extract the downloaded 3DNA application to Cygwin's home directory.
4. Execute the Cygwin terminal
5. Navigate to the location where the PDB files are located
6. Execute command "find -type f | xargs -i ~/X3DNA/bin/find_pair {} {}.inp"
7. Execute command "find -type f -name "*.inp" | xargs -i ~/X3DNA/bin/analyze {}"

Installation and usage on MinGW/MSYS is the same.

6.1.3.1.2 The Result of 3DNA

The SETTER (and also MultiSETTER) uses the *.out files. For us the most important information from the file is the list of Watson-Crick bonds between base-pairs.

6.2 Dependencies

As the Engine was developed in Microsoft Visual Studio 2010, the executable application is compatible with Microsoft Windows operating systems. The application was tested both on 32 and 64 bit versions of Microsoft Windows 7.

To run the application on a computer that does not have Visual C++ 2010 installed, Microsoft Visual C++ 2010 Redistributable Package must be installed.

Since Java is platform independent, the GUI could be executed on any systems with installed Java Virtual Machine. However, as the Engine is dependent on Windows and the GUI uses external process call to execute the Engine, GUI is also dependent on Windows systems.

To execute the GUI, Java Runtime Environment 6 must be installed.

6.3 Java GUI

6.3.1 Executing the GUI

To execute the GUI, simply open the MultiSETTER application. At the first execution a dialog will appear that informs you to select the Engine application.

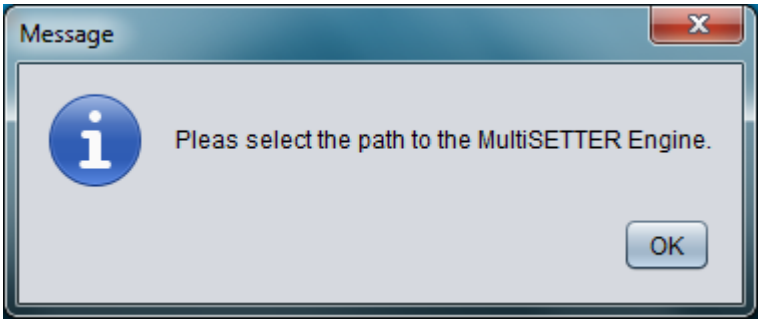


Figure 15: Warning dialog to select the MultiSETTER executable

After clicking on the "OK" button the Settings window will show up where you can select the executable MultiSETTER Engine.

6.3.2 Settings Window

This window is used to specify the path to the Engine application and also to set the MultiSETTER algorithm's parameters.

MultiSETTER Engine

While the path to the Engine is not specified the GUI has only limited functionality. Click on the highlighted button. A file chooser dialog will open. Generally, this type of button opens a file chooser dialog.

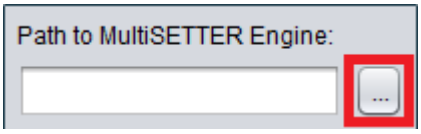


Figure 16: Button to open file chooser dialog

You will be informed if the selected executable is not the MultiSETTER Engine.

Number of Threads

You can specify the number of working threads used by the Engine. Be careful when selecting, the parallelization is more effective, if the input RNA structures are large, it can slow your computer down.

Algorithms Parameters

As it has been introduced, the algorithm has a few parameters:

- NJ Algorithm - two Neighbor Joining methods are implemented, a standard and a modified method that recalculates the guide tree after all merging.
- Merging Algorithm - two algorithms are implemented to select the residue at merging.
- Since the previous version of SETTER was significantly not symmetric, in the manner, that the distance between structure A and structure B was different, then the distance between structure B and structure A, a parameter was added to resolve this issue. This parameter determines, how to calculate symmetric distance from two distances. The possibilities are the following: choose the smaller distance, calculate arithmetic mean or calculate geometric mean. However, the actual version of SETTER is almost symmetric, therefore in most cases this parameter has insignificant effect on the results.
- Parameters to reduce the effect of outliers.

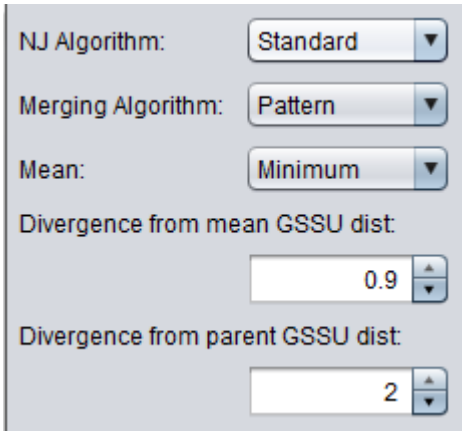


Figure 17: The algorithm's settings

Printing to PDB file

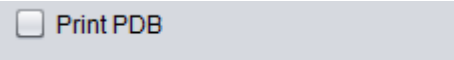


Figure 18: PDB file printing

This checkbox enables printing the average RNA structures and input structures to PDB files. The printing is available only if a set of RNA structures is merged to average RNA. PDBs created from input RNAs will be superposed to the average RNA structure. The output PDB files will be located in the same directory as the average RNA. The name of average PDB is derived from the name of the input file. The name of the PDB files containing the superposed RNA structures are composed from input family name, PDB ID, and chain ID according the following pattern: [input name]_[PDB ID]-[chain ID].

6.3.3 Importing RNA Structures

As previously mentioned, the SETTER is using PDB and 3DNA files as an input. To add those files to the application's database, click on the button "Add RNA to DB"! This will open a dialog:

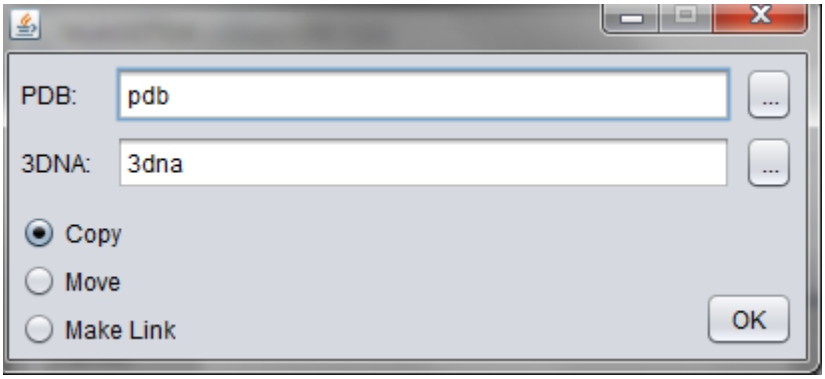


Figure 19: Importing RNA to MultiSETTER's database

Here you can select the PDB and 3DNA files as well as the import method. There is a restriction, the two input files belonging to one PDB ID (respectively to RNA structure) have to have the same name.

6.3.4 Create Input File

To create the input file, click on "Create Input File", "Create Family" or "Create Test" buttons. The default behavior of "Create Family" and "Create Test" buttons are creating

the input file in *GUI/data/temp* directory. The temporary files will not be deleted, only rewritten.

After clicking one of the above mentioned button, a window will pop up:

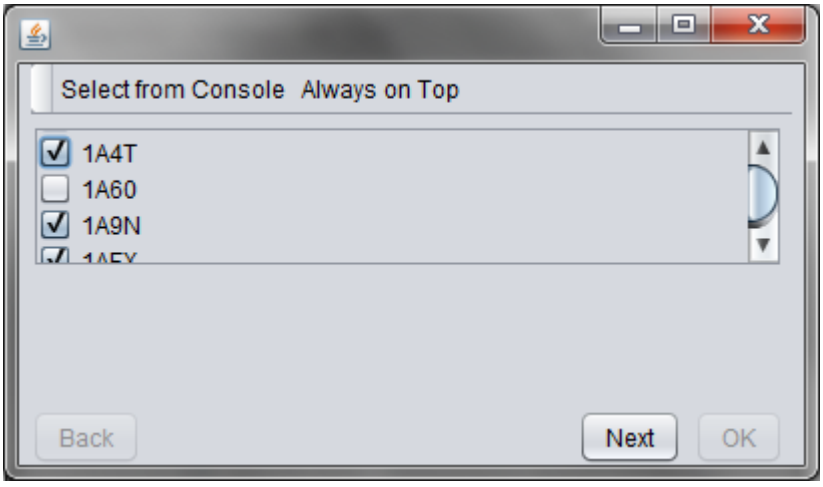


Figure 20: Selecting the input PDB files

The window has an "Always on Top" option that could be very useful in situations when the user reads the list of input RNAs from an Internet browser or from file.

Here you can manually select PDB IDs or you can click on "Select from Console". The console parses PDB IDs from textual input and selects PDB structures presented in the application's database:

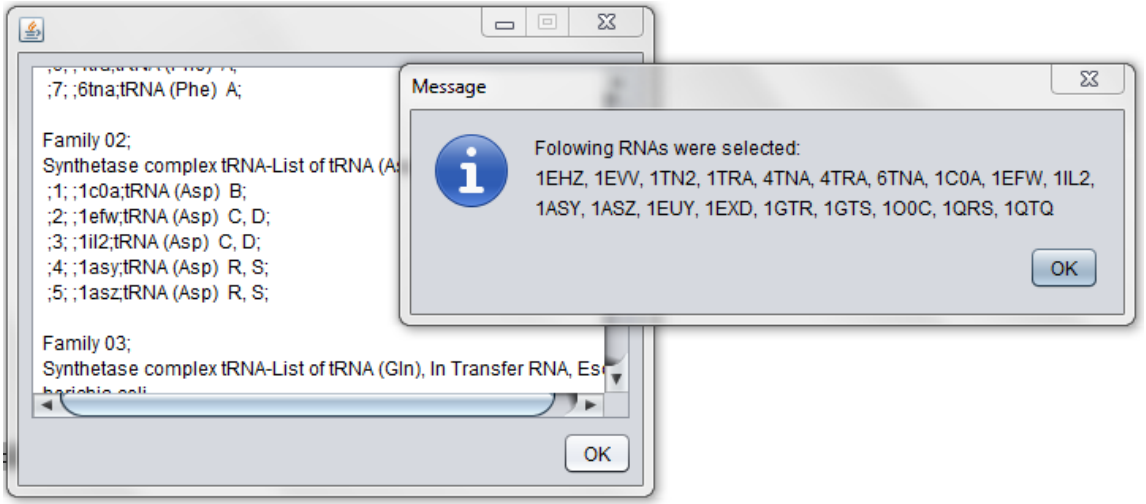


Figure 21: Parsing PDB identifiers from textual

By clicking the “Next” button you can select the chain IDs from the PDB file:

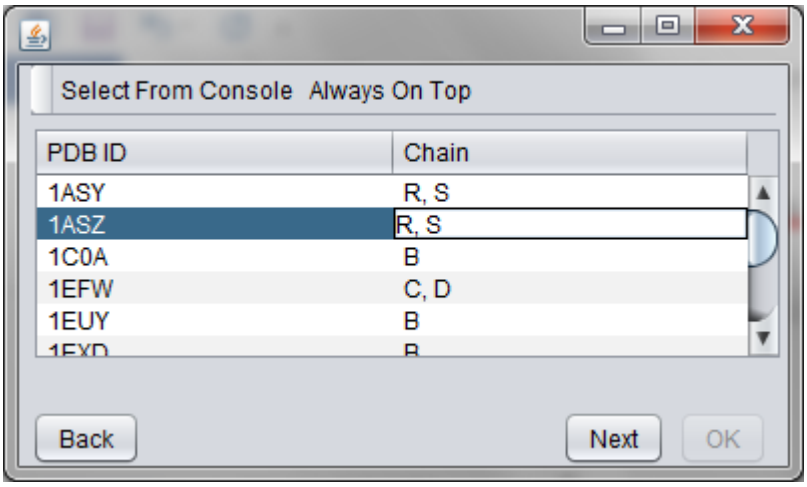


Figure 22: Selecting the chain identifiers

The application offers you all the chains from PDB, which represents RNA or DNA structure. The chains are editable by clicking. The only permitted separator between chain IDs is comma “,”.

By clicking the “Next” button you can specify a description for the input set of RNA structures.

6.3.5 Creating Average RNA

To create the average RNA, please select an input file or a set of input files. By clicking the “Start” button a file chooser dialog will pop up. Here you can specify the name and location of the serialized average RNA structure. If multiple input files were selected, only the location of serialized structures can be chosen, the name of structures will be derived from the name of input family.

The textual output contains the location of input and output files, the mean distance of input RNA structures from the average RNA structure and the deviation.

If the “Print PDB” checkbox is checked in the settings window, the output will contain a Jmol script to visualization the results of the merging process.

The textual output could be saved by clicking on the “Save” button.

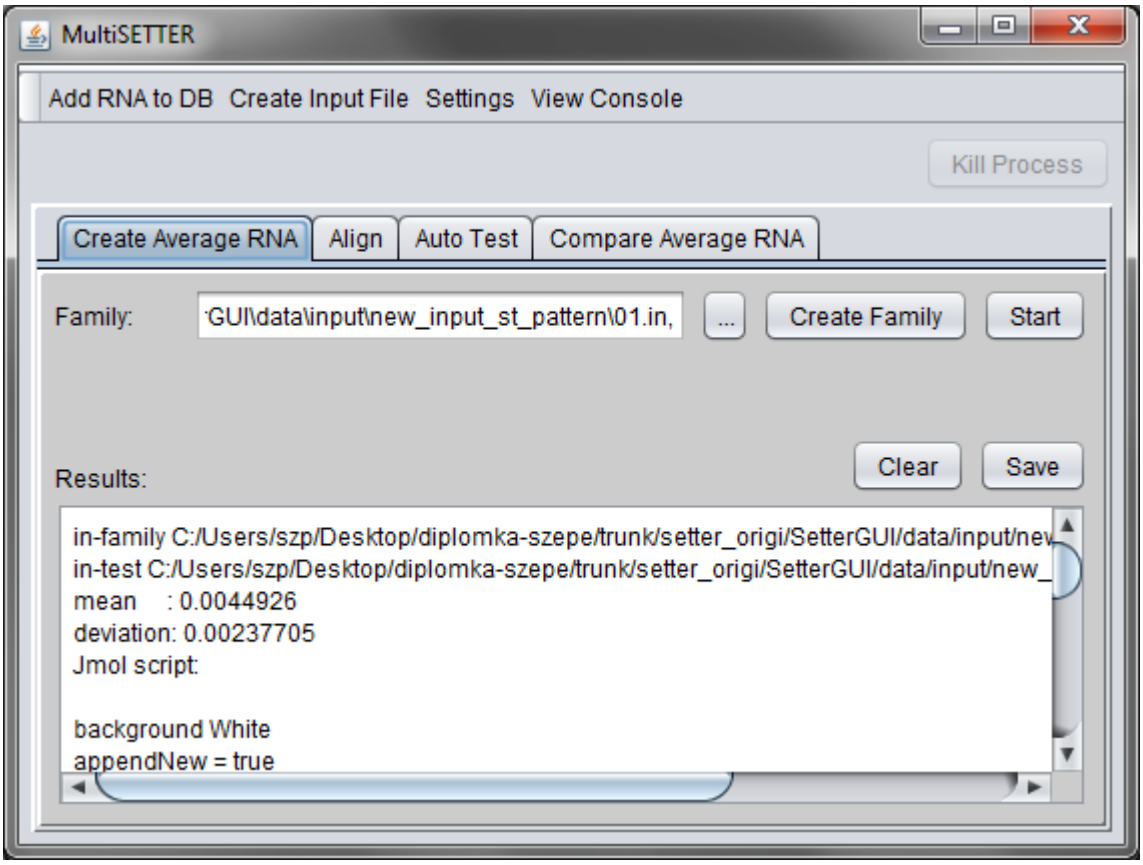


Figure 23: Creating average RNA from a family

6.3.6 Aligning Set of RNA with a Family

There are multiple ways to align RNA to a family represented by an average RNA:

- The test family could be an input file (*.in) or a serialized average RNA (*.aver).
- The tested set of RNA structures can be only input file (*.in).
- Multiple inputs are supported.

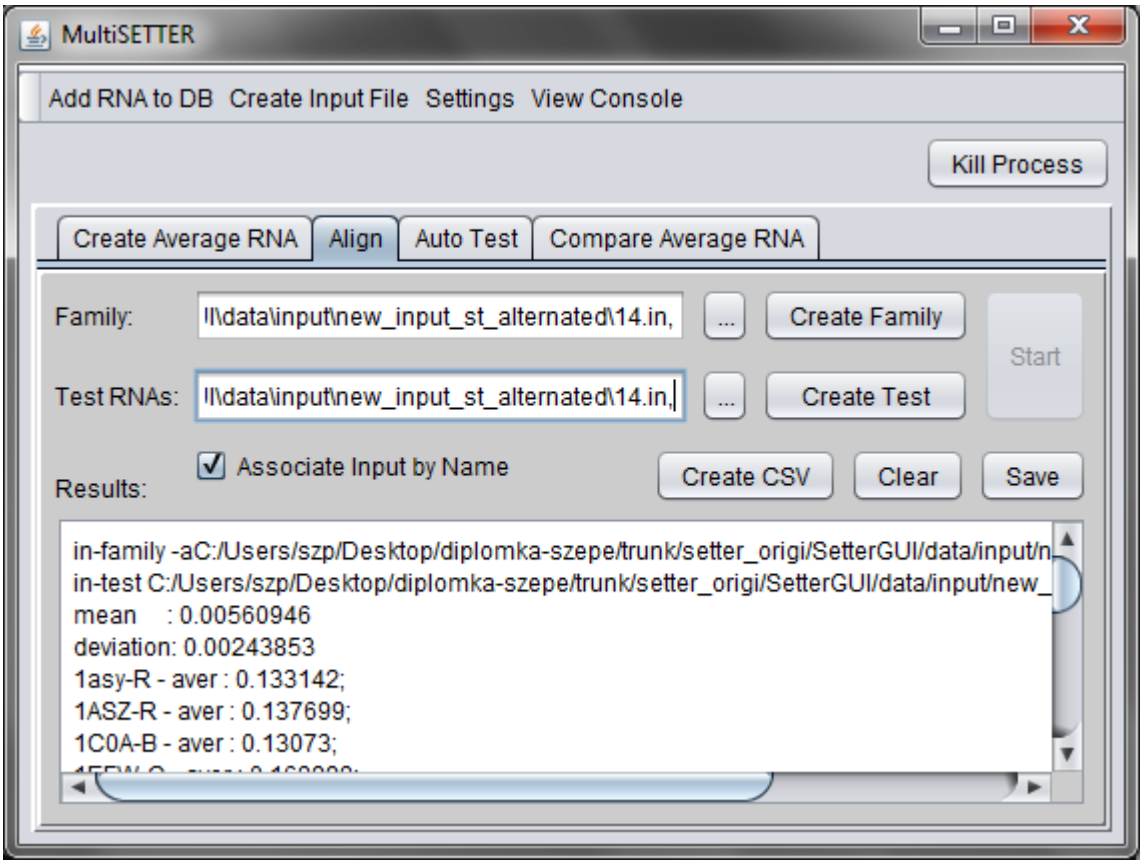


Figure 24: Aligning set of RNA structures with average RNA

The default behavior of aligning in case when multiple inputs are selected, is to associate input files based on theirs name; which means that the application would work with input files with same name but different extension, as they would represent the same set of RNA structures. Consider the following situation: input families contain files *family.in*, *family.aver* and the test RNAs contain *family.in*. The program will not align the test set *family.in* with *family.aver* and the program will not create average RNA from the whole *family.in*. The behavior will be the following: the Engine takes the RNA structures from the test set one by one, checks if the RNA is present in the input family set; if it is, the average RNA will be constructed from the input set without the actually tested RNA.

To avoid this behavior, uncheck the “Associate Input by Name” checkbox.

The textual output contains the input files, the mean distance of input RNA structures from the average RNA structure, the deviation and finally, the distances of test RNAs from the family represented by an average RNA.

The “Create CSV” button will allow you to create a CSV file that contains distances from tested RNA structures from each input family.

6.3.7 Automated Testing

This option is to test the parameters of the algorithm. The input folder is the folder where input files (*.in) are located; this folder will also be used to store average RNA structures. The output folder is used to store the generated CSV files.

6.3.8 Comparing Two Average RNA

This option is to compare – mainly visually – two RNA structures. The result of this comparison will be the distance between structures computed by SETTER and a Jmol script.

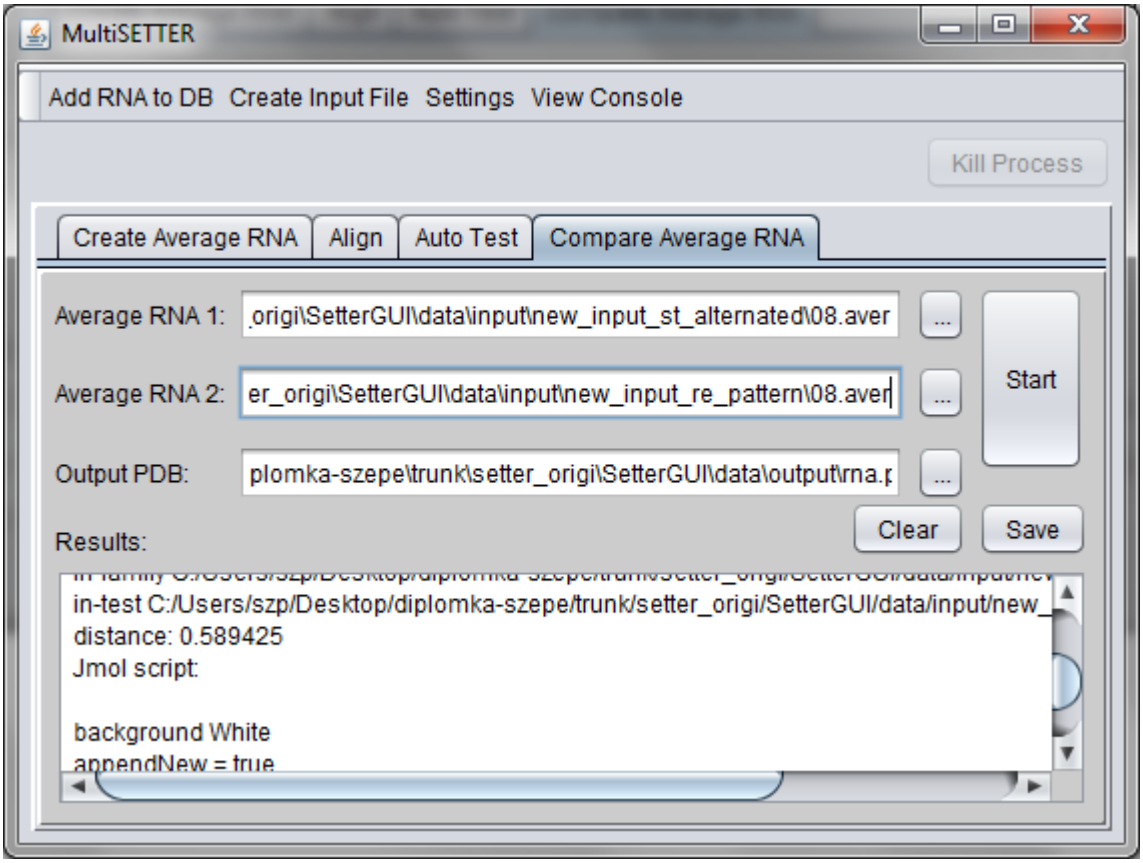


Figure 25: Comparing two average RNA structures

6.4 Jmol

Jmol is an open-source Java viewer for chemical structures in 3Dwith features for chemicals, crystals, materials and biomolecules. The application can be downloaded from: http://jmol.sourceforge.net/

This application can be used to visualize the PDB file results of RNA family merging. After downloading and opening the application, click on File → Console and copy the Jmol script printed to output.

```
background White
appendNew = true
load C:/MultiSETTER/GUI/data/input/families/12.pdb
load append C:/MultiSETTER/GUI/data/input/families/12_1CSL-A.pdb
load append C:/MultiSETTER/GUI/data/input/families/12_1duq-A.pdb
load append C:/MultiSETTER/GUI/data/input/families/12_1EBQ-A.pdb
load append C:/MultiSETTER/GUI/data/input/families/12_1EBR-A.pdb
load append C:/MultiSETTER/GUI/data/input/families/12_1EBS-A.pdb
load append C:/MultiSETTER/GUI/data/input/families/12_1ETF-A.pdb
load append C:/MultiSETTER/GUI/data/input/families/12_1i9f-A.pdb
select all; trace only;
select */2; color trace Red
select */3; color trace Orange
select */4; color trace Yellow
select */5; color trace Green
select */6; color trace Cyan
select */7; color trace Blue
select */8; color trace Indigo
select */1; color trace Black
```

This script visualizes the results, where average RNA has black color.

To navigate between RNAs, use the arrow buttons:

Figure 26: Navigating between structures in Jmol

To show all RNA use the highlighted button:

Figure 27: Show all structures in Jmol

7 Results and Consequences

The MultiSETTER algorithm was compared with SETTER on a dataset containing 14 functionally correlating RNA families (see 13.1). The algorithms are compared in two aspects: the quality of the algorithm and the required time.

For testing a Lenovo ThinkPad T420 has been used with Intel Core i7-2620M processor, 4 Gigabytes of RAM and hard drive with 7200 rpm; running on Microsoft Windows 7, 64 bit. The processor has 2 cores and 4 threads.

7.1 Quality Measurement

7.1.1 MultiSETTER's Results

To determine the closest family to an RNA, average RNA structures are computed from each family. The distance is calculated between the RNA and the average RNA represented the family. In order to test the algorithm properly, the average RNA of the family, where the actually tested RNA belongs, is calculated without taking into consideration the actually tested RNA. The result of the single RNA-to-all family alignment is an ordered list of families; sorted by the distances from the tested RNA. The important data is the position of family, where the RNA belongs, in the ordered list. The quality of alignment method is determined by the percentage of above described positions.

7.1.1.1 Settings

The MultiSETTER algorithm has been tested with two settings:

- "Standard" Neighbor-Joining method, "Alternating" merging algorithm, Divergence from mean: 0.9, Divergence from parent: 2.0 – in chart and tables marked as **Alternating**.
- "Standard" Neighbor-Joining method, "Pattern" merging algorithm, Divergence from mean: 0.8, Divergence from parent: 1.4 – in chart and tables marked as **Pattern**.

7.1.2 SETTER's Results

Since the SETTER is a pairwise alignment algorithm, it is hard to compare with a multiple alignment tool, therefore the result of all-to-all pairwise alignment must be processed to correspond to the above described results.

Two methods were used:

- The results of one-to-all comparisons are arranged by the achieved distance. The order of families is calculated based on the first occurrence of family's member in the sorted list. Let's consider that the data is represented in the way [the family's ID where the actual RNA belongs to; distance]. The sorted list is [1; 0.12], [1; 0.2], [3; 0.32], [2; 0.55], [1; 0.7], [4; 1.12], [6; 1.25], [1; 2.13], [5; 3.14]; the ordered list of families is 1, 3, 2, 4, 6, 5 – in chart and tables marked as **Best**.
- The mean distance is calculated between the actually tested RNA and the families. Let's consider the same data presented above. The mean distance to family with ID 1 is (0.12 + 0.2 + 0.7 + 2.13) / 4 = 0.7875; from family with ID 2 is 0.55; from 3 is 0.32; from 4 is 1.12; from 5 is 3.14; from 6 is 1.25; so, the ordered list of families is 3, 2, 1, 4, 6, 5 – in chart and tables marked as **Mean**.

7.1.3 Comparison

The following table contains the distribution of the above introduced positions for each test cases:

Position	Alternating	Pattern	Best	Mean
1	79.38%	78.35%	86.60%	70.10%
2	7.22%	2.06%	2.06%	4.12%
3	3.09%	5.15%	5.15%	2.06%
4	1.03%	3.09%	0.00%	6.19%
5	1.03%	2.06%	1.03%	6.19%
6	1.03%	1.03%	0.00%	2.06%
7	1.03%	2.06%	2.06%	0.00%
8	1.03%	0%	0.00%	2.06%
9	0%	0%	2.06%	1.03%
10	2.06%	2.06%	0.00%	1.03%
11	2.06%	3.09%	0.00%	1.03%
12	1.03%	0%	0.00%	1.03%
13	0%	0%	1.03%	3.09%
14	0%	1.03%	0.00%	0.00%

Table 1: Comparison with SETTER

7.1.4 The acquired results broken down for families

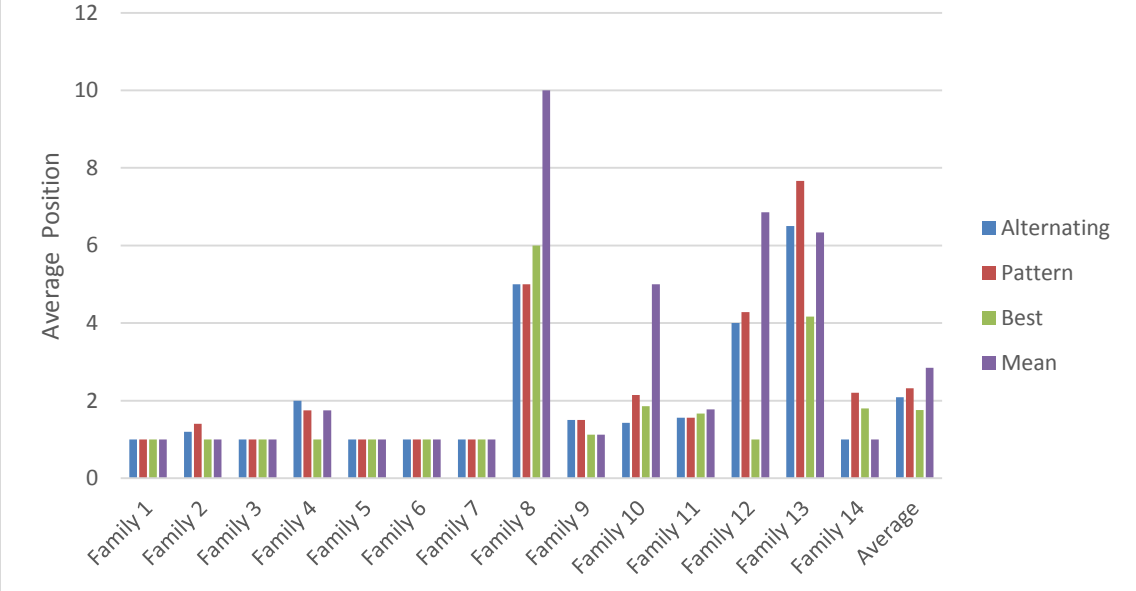


Figure 28: Comparison with SETTER, average positions for families

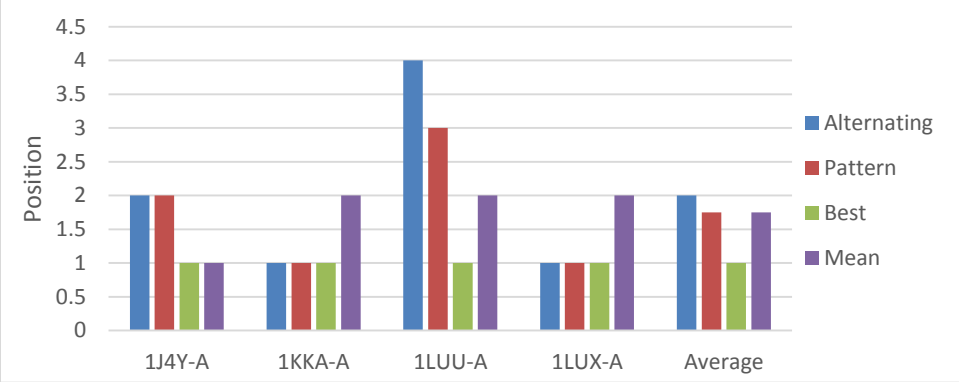


Figure 29: Comparison with SETTER - Family 4

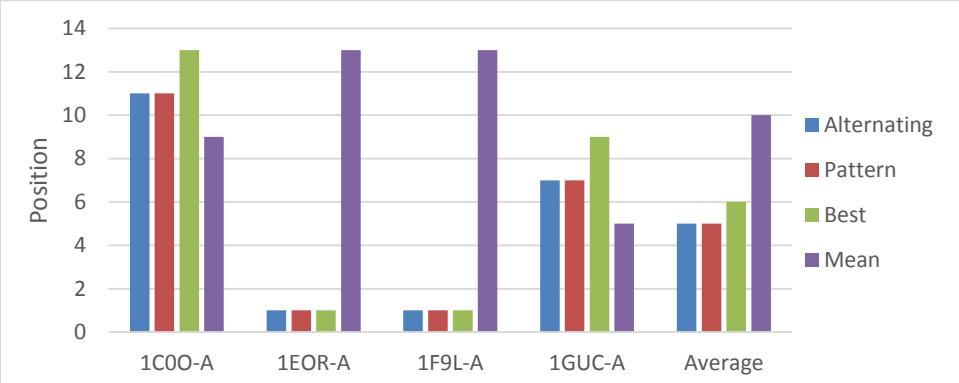


Figure 30: Comparison with SETTER - Family 8

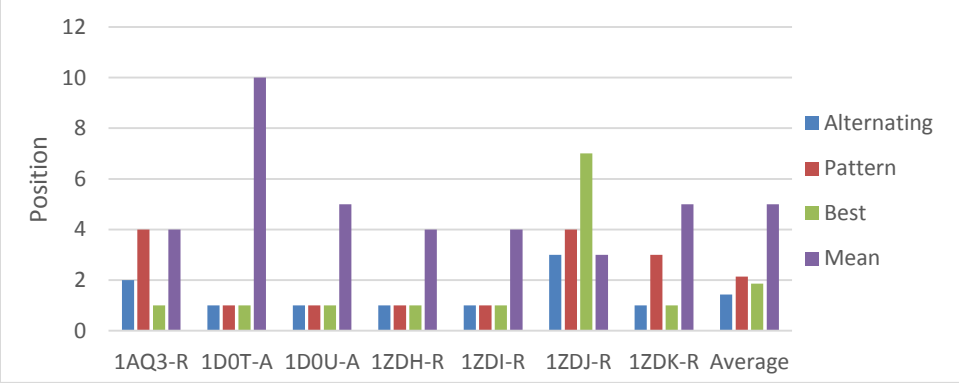


Figure 31: Comparison with SETTER - Family 10

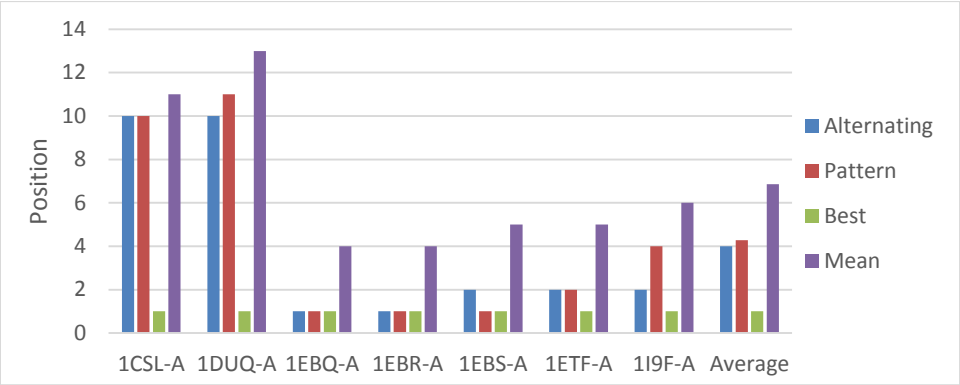


Figure 32: Comparison with SETTER - Family 12

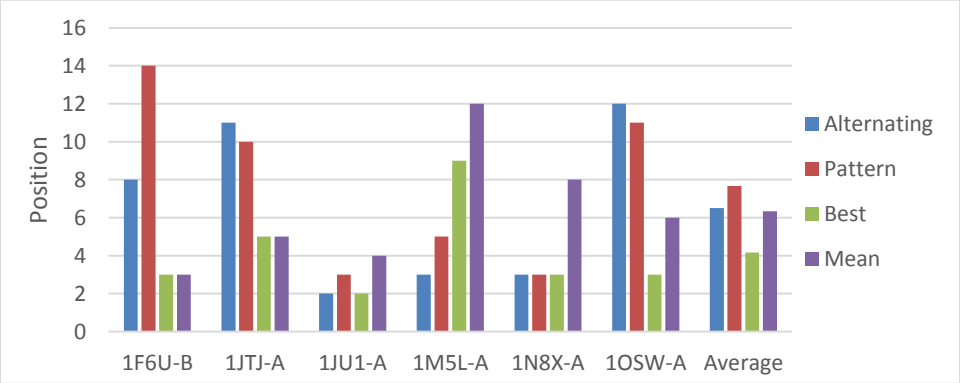


Figure 33: Comparison with SETTER - Family 13

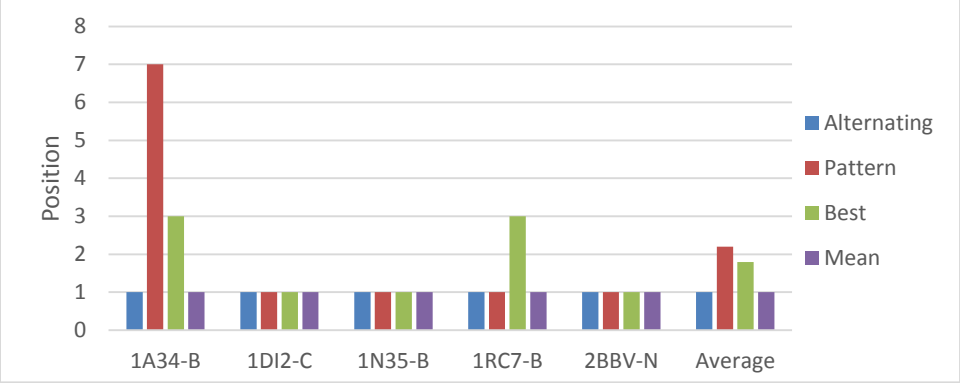


Figure 34: Comparison with SETTER - Family 14

As we can see in the charts above, the quality of multiple alignment is comparable with the SETTER's result, however some graphs demonstrate that the concept of multiple alignment – respectively the merging of three dimensional structure – for determining the

functionality is a better solution. A perfect example for this scenario would be the Family 14 (see Figure 34).

In some cases the worst precision of multiple alignment can be explained by a fact, that the family contains functionally correlated but structurally less correlated RNA structures. In such event it is easier to find one similar structure, than to generate an average structure that is similar to all structures in the family. An example to illustrate this situation would be Family 12 (see Figure 32). The mean distance of each RNA from the other RNAs in the family is relatively big, however all the RNAs have at least one RNA in the family that is close.

Family 12	1CSL-A	1DUQ-A	1EBQ-A	1EBR-A	1EBS-A	1ETF-A	1I9F-A
1CSL-A	NA	0.086218	3.12224	5.04567	4.34906	4.78887	9.09866
1DUQ-A	0.086218	NA	4.33376	5.35448	5.33012	5.61413	13.1036
1EBQ-A	3.12224	4.33376	NA	0.358163	0.195691	0.978579	1.76286
1EBR-A	5.04567	5.35448	0.358163	NA	0.557735	0.803556	1.0327
1EBS-A	4.34906	5.33012	0.195691	0.557735	NA	0.595451	2.45556
1ETF-A	4.78887	5.61413	0.978579	0.803556	0.595451	NA	0.238314
1I9F-A	9.09866	13.1036	1.76286	1.0327	2.45556	0.238314	NA

Table 2: All-to-all distances in family 12

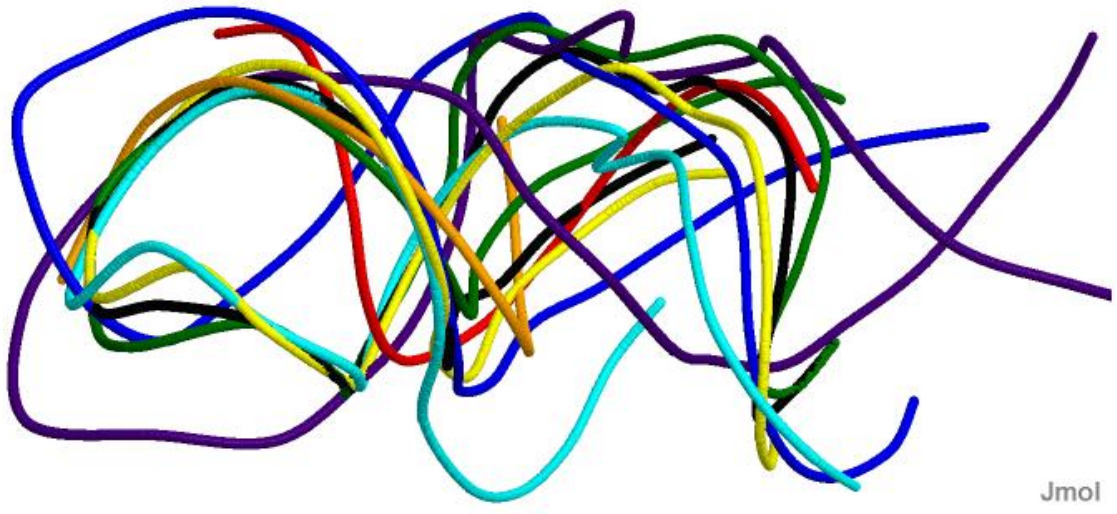


Figure 35: Family 12; Average RNA is black, 1CSL-A is red, 1DUQ-A is orange, 1EBQ-A is yellow, 1EBR-A is green, 1EBS-A is cyan, 1ETF-A is blue and 1I9F-A is indigo.

7.2 Computation Time

Since parsing the PDB files is very time-consuming, the calculation time comparison will be presented with and without the required time to read the RNA structures. Since MultiSETTER is multithreaded, the time will be presented using single thread and also using four threads. The time is measured in seconds.

7.2.1 Creating Families

The MultiSETTER algorithm aligns RNA structures to average RNA structures representing the whole families, therefore it is required to create the average structures. Since this is a one-time task, the required time to create average structures will not be included in the time required to aligning all RNA from dataset.

Family	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**All**
1 Thread	7.9	3.4	8.0	0.4	0.8	0.8	114.2	0.4	177.0	1.1	11.6	15.4	2.6	1.6	345.3
4 Threads	4.9	2.0	4.7	0.3	0.4	0.5	57.7	0.2	108.2	0.7	5.9	9.9	1.5	1.0	197.8
Speedup	1.62	1.72	1.70	1.73	2.04	1.65	1.98	1.81	1.64	1.60	1.96	1.56	1.78	1.58	1.75

Table 3: Required time to create families

7.2.2 Comparison with SETTER

Application	**Method**	**Time**
SETTER	Normal	14058.3
	Buffered	1733.6
	Without Reading	1669.1
MultiSETTER - 1 Thread	Including Create Average RNA	1607.9
	Normal	1262.6
	Without Reading	359.2
MultiSETTER - 4 Threads	Including Create Average RNA	713.6
	Normal	515.8
	Without Reading	159.6

Table 4: Required time to compare RNA structures with families

While testing the SETTER using the Buffered method all the RNA structures were read at the beginning of the alignment.

We can read from the table above that the MultiSETTER application brings a big speedup in contrast to the SETTER application.

7.3 MultiSETTER's Parameters

As it was mentioned there are implemented two methods for neighbor-joining and two methods for residue selection. In addition there are two parameters, the allowed divergence of the distance of paired GSSUs from the mean distance and the allowed divergence of merged GSSU from its parents.

7.3.1.1.1 The Best Parameters

Neighbor-Joining Method:	Standard		Recalculated	
Residue Selection Method:	Alternating	Pattern	Alternating	Pattern
Divergence from Mean:	0.9	0.8	0.5	0.5
Divergence from Parents:	2.0	1.4	2.5	3.0

Table 5: Chosen parameters

7.3.1.1.2 Results

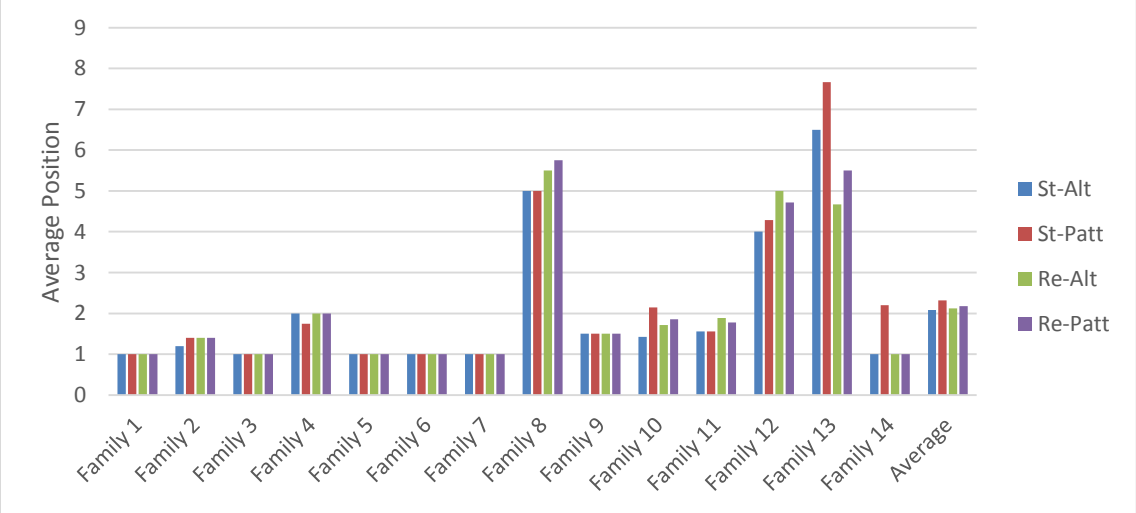


Figure 36: Comparison of MultiSETTER's parameters

Neighbor-Joining Method:	Standard		Recalculated	
Residue Selection Method:	Alternating	Pattern	Alternating	Pattern
Position 1	79.38%	78.35%	78.35%	81.44%
Position 2	7.22%	2.06%	7.22%	4.12%
Position 3	3.09%	5.15%	4.12%	2.06%
Position 4	1.03%	3.09%	3.09%	3.09%
Position 5	1.03%	2.06%	1.03%	2.06%
Position 6	1.03%	1.03%	0%	1.03%
Position 7	1.03%	2.06%	0%	0%
Position 8	1.03%	0%	1.03%	1.03%
Position 9	0%	0%	1.03%	1.03%
Position 10	2.06%	2.06%	1.03%	0%
Position 11	2.06%	3.09%	0%	0%
Position 12	1.03%	0%	0%	1.03%
Position 13	0%	0%	0%	0%
Position 14	0%	1.03%	3.09%	3.09%

Table 6: Comparison of MultiSETTER's parameters

As we can see, it is not clear which are the best settings for the MultiSETER algorithm.

7.3.1.1.3 Differences in Average RNA Structures

While the Alternating method seems more accurate, the average structure generated by Pattern method is closer to reel world RNA structures, it seems more continuous.

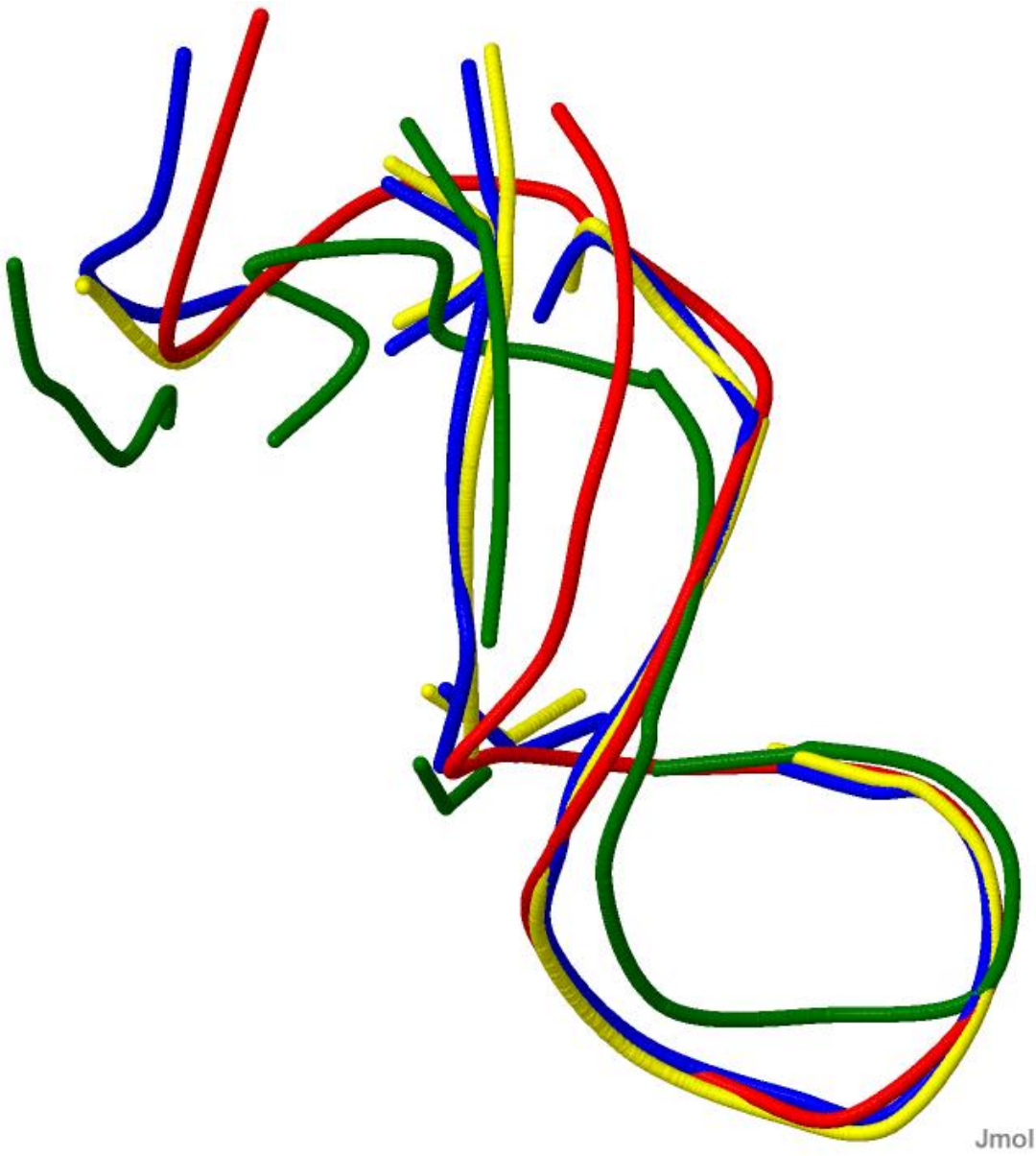


Figure 37: Average RNA structures generated from family 12; Standard-Pattern is red, Standard-Alternating is blue, Recalculate-Pattern is green and Recalculate-Alternating is yellow.

8 Future Work

8.1 Optimize for Families

Since there are no two families with same structures, there are no globally optional parameters for the MultiSETTER algorithm. The families should be considered as isolated units and the optimal parameters should be chosen for each family and not for the whole algorithm. Computationally that could be done by writing a module to GUI, which tests the parameters for one family. The goal should be to minimize the distance from each RNA to the average RNA calculated from the rest of structures in the family. To get optimal results, it would be rational to let the users do the final decision by reviewing the visualized results.

To facilitate this, a module should be integrated to the GUI, to visualize the results.

8.2 Distributed System

It would be useful to re implement the GUI application as a part of distributed system.

Since the size of PDB files is usually less than a megabyte and but in extreme cases it reach more than 5 megabyte, the data transfer could be the most significant problem. To reduce the size of transferred data, it is recommended to transfer serialized RNA structures, which contains only necessary information. Communication between Java Server and Clients could be implemented using JMS (Java Message Service) as point-to-point messaging system.

The other possible solution to distributing the input RNA structures could be a shared data storage, where all the clients have access.

9 Bibliography

RSCB Protein Data Bank. [Online] RCSB Protein Data Bank. http://www.pdb.org/.

Berg, Jeremy M., Tymoczko, John L. and Stryer, Lubert. 2002. *Biochemistry, 5th edition.* New York : s.n., 2002.

CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, osition-specific gap penalties and weight matrix choice. **Thompson, Julie D., Higgins, Desmond G. and Gibson, Toby J. 1994.** 1994, Nucleic Acids Research, Vol. 22, pp. 4673-4680.

Efficient RNA pairwise structure comparison by SETTER method. **Hoksza, David and Svozil, Daniel. 2011.** 2011, BIOINFORMATICS, pp. 1-7.

Hoksza, David and Svozil, Daniel. 2011. SETTER - RNA SEcondary sTructure-based TERtiary Structure Similarity Algorithm, presentation. *ISBRA.* Changsha, China : LNBI, Springer Verlag, 2011.

Human Molecular Genetics, 2006, Vol. 15, Review Issue 1. **Mattick, John S. and Makunin, Igor V. . 2006.** 2006, Oxford Journals.

Jones, Neil C. and Pevzner, Pavel A. 2004. *An Introduction to Bioinformatics Algorithms.* Massachusetts : Massachusetts Institute of Technology, 2004.

Joyce, Gerald F. and Orgel, Leslie E. 1993. Prospects for Understanding the Origin of the RNA World. *The RNA World.* 1993, pp. 1-25.

Paradigms for computational nucleic acid design. **Dirks, Robert M., et al. 2004.** 2004, Nucleic Acids Research.

Tamura, Makio , et al. 2004. Structural Slissification of RNA. [Online] Lawrence Berkeley National Laboratory, 2004. http://scor.berkeley.edu/.

The Neighbor-joining Method: A New Method for Reconstructing Phylogenetic Trees. **Saitou, Naruya and Nei, Masatoshi. 1987.** 1987, Molecular Biology and Evolution, pp. 406-425.

Westhof, Eric and Auffinge, Pascal . 2000. RNA Tertiary Structure. *Encyclopedia of Analytical Chemistry.* Chichester : John Wiley & Sons Ltd, Chichester, 2000, pp. 5222–5232.

10 List of Figures

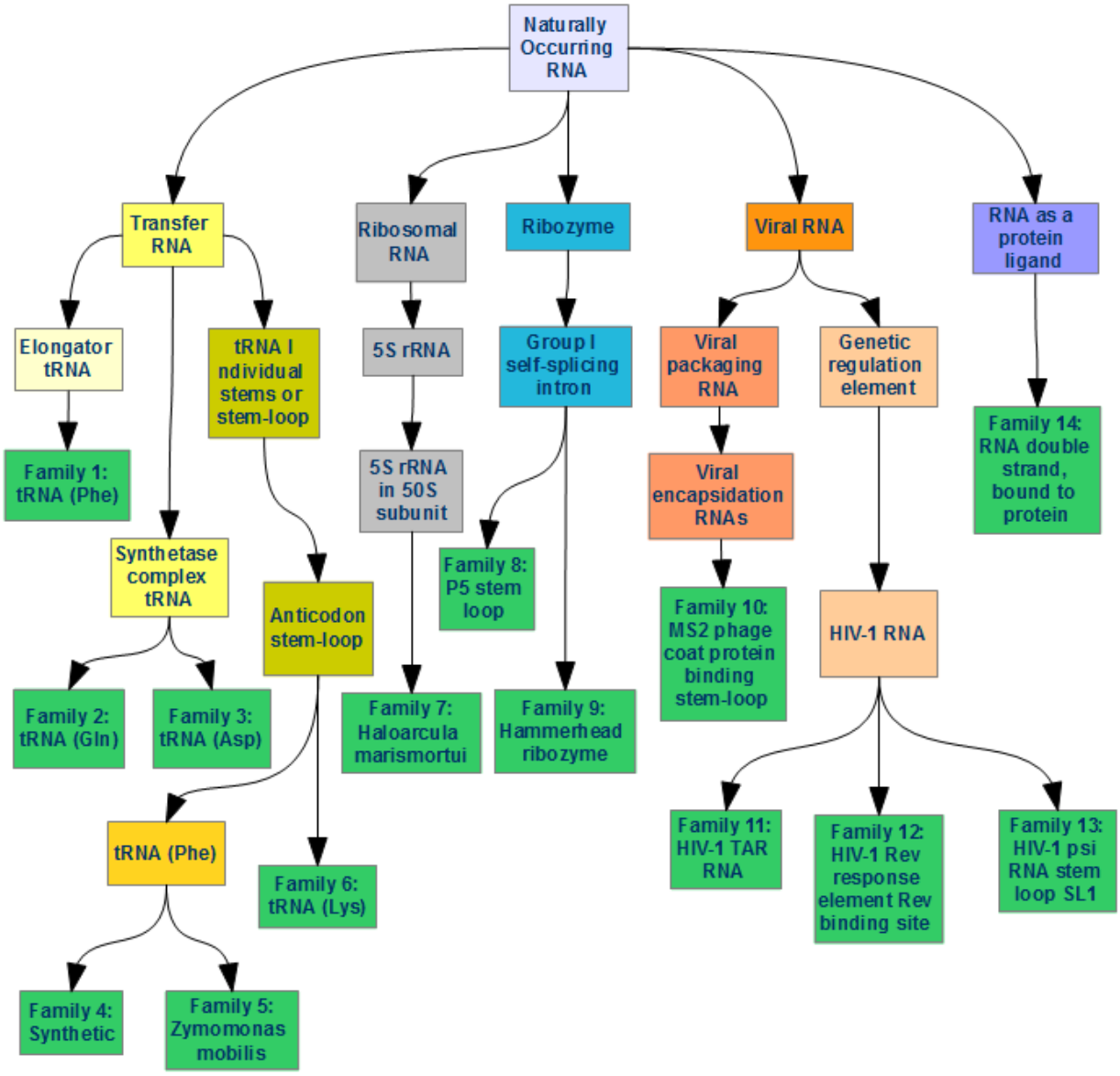
11 List of Tables

12 List of Abbreviations

SETTER	SEcondary sTructure-based TERtiary Structure Similarity Algorithm
RNA	Ribonucleic acid
DNA	Deoxyribonucleic acid
A	Adenine
G	Guanine
C	Cytosine
T	Thymine
U	Uracil
mRNA	Messenger RNA
tRNA	Transfer RNA
rRNA	Ribosomal RNA
ncRNA	Non-coding RNA
HIV	Human immunodeficiency virus
GSSU	Generalized secondary structure units
PDB	Protein Data Bank
TBB	Intel Thread Building Blocks
BOOST	C++ libraries
3DNA	Software tool, in some case a reference for file generated by 3DNA
Jmol	Software for visualization

13 Appendix

13.1 Input Families



Family 1: 1EHZ - A, 1EVV - A, 1TN2 - A, 1TRA - A, 4TNA - A, 4TRA - A, 6TNA - A

Family 2: 1C0A - B, 1EFW - C, 1IL2 - C, 1ASY - R, 1ASZ - R

Family 3: 1EUY - B, 1EXD - B, 1GTR - B, 1GTS - B, 1O0C - B, 1QRS - B, 1QTQ - B

Family 4: 1J4Y - A, 1KKA - A, 1LUU - A, 1LUX - A

Family 5: 1Q2R - E, 1Q2R - F, 1Q2S - E, 1Q2S - F

Family 6: 1BZ2 - A, 1BZ3 - A, 1BZT - A, 1BZU - A, 1FEQ - A, 1FL8 - A

Family 7: 1JJ2 - 9, 1K73 - B, 1K8A - B, 1K9M - B, 1KC8 - B, 1KD1 - B, 1KQS - 9, 1M1K - B, 1M90 - B, 1N8R - B, 1NJI - B, 1Q7Y - B, 1Q81 - B, 1Q82 - B, 1Q86 - B, 1QVF - 9, 1QVG - 9, 1S72 - 9

Family 8: 1C0O - A, 1EOR - A, 1F9L - A, 1GUC - A

Family 9: 1NYI - A, 1Q29 - A, 1HMH - A, 1MME - A, 299D - A, 359D - A, 379D - A, 488D - A

Family 10: 1AQ3 - A, 1D0T - A, 1D0U - A, 1DZS - A, 1GKV - R, 1GKW - R, 1H8J - R, 1KUO - R, 1ZDH - A, 1ZDI - A, 1ZDJ - A, 1ZDK - A

Family 11: 1ANR - A, 1ARJ - A, 1QD3 - A, 397D - A, 1AKX - A, 1LVJ - A, 1UTS - B, 1UUD - B, 1UUI - A

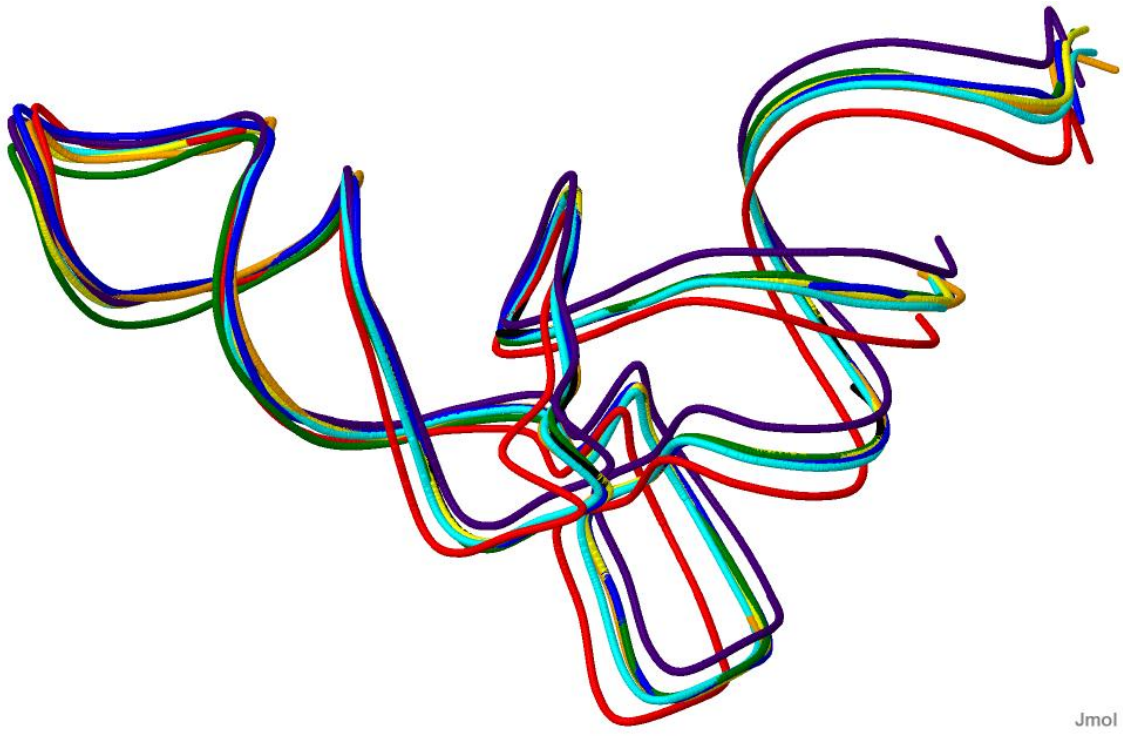
Family 12: 1CSL - A, 1DUQ - A, 1EBQ - A, 1EBR - A, 1EBS - A, 1ETF - A, 1I9F - A

Family 13: 1M5L - A, 1N8X - A, 1JTJ - A, 1JU1 - A, 1F6U - A, 1OSW - A

Family 14: 1A34 - A, 2BBV - A, 1RC7 - A, 1DI2 - A, 1N35 – A

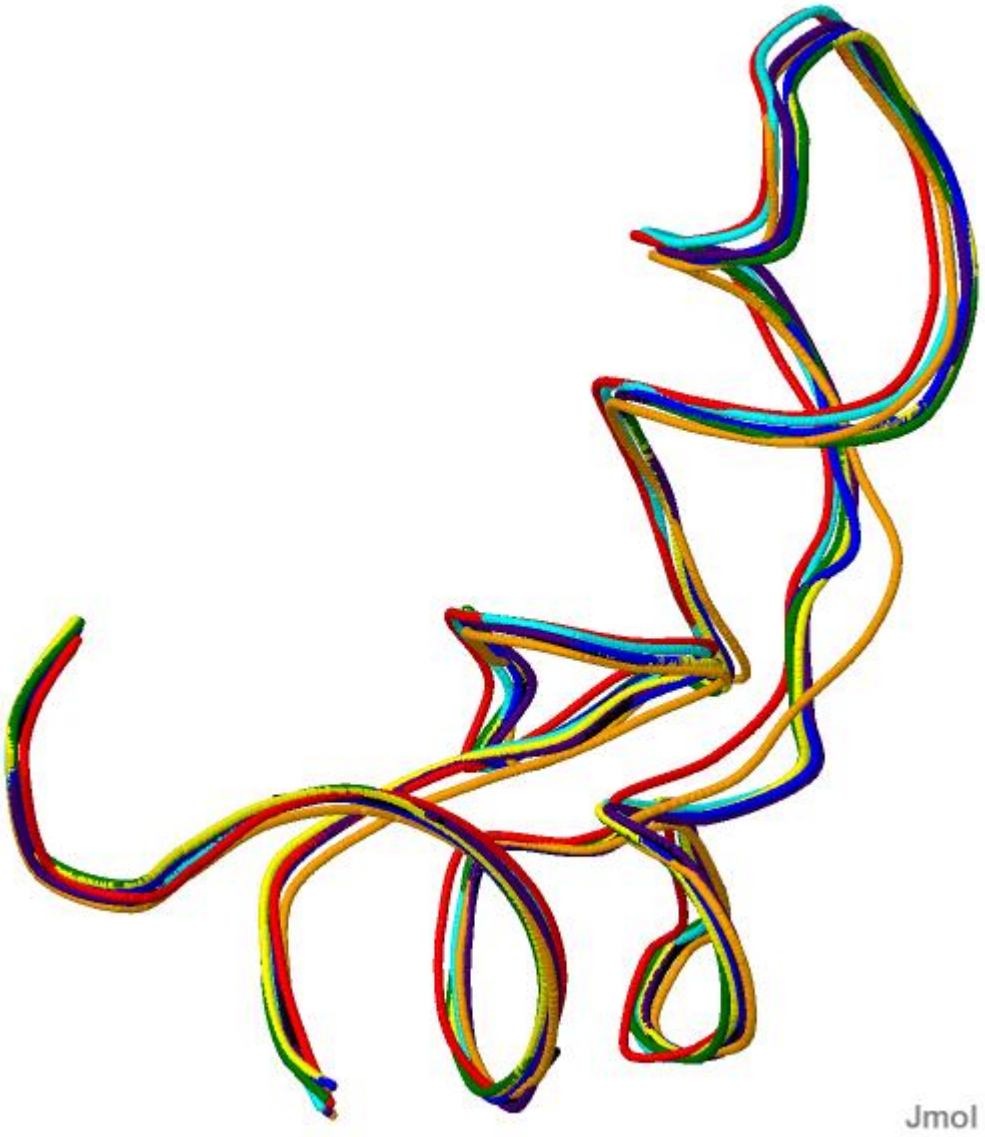
13.2 3D alignments

13.2.1 Family 1

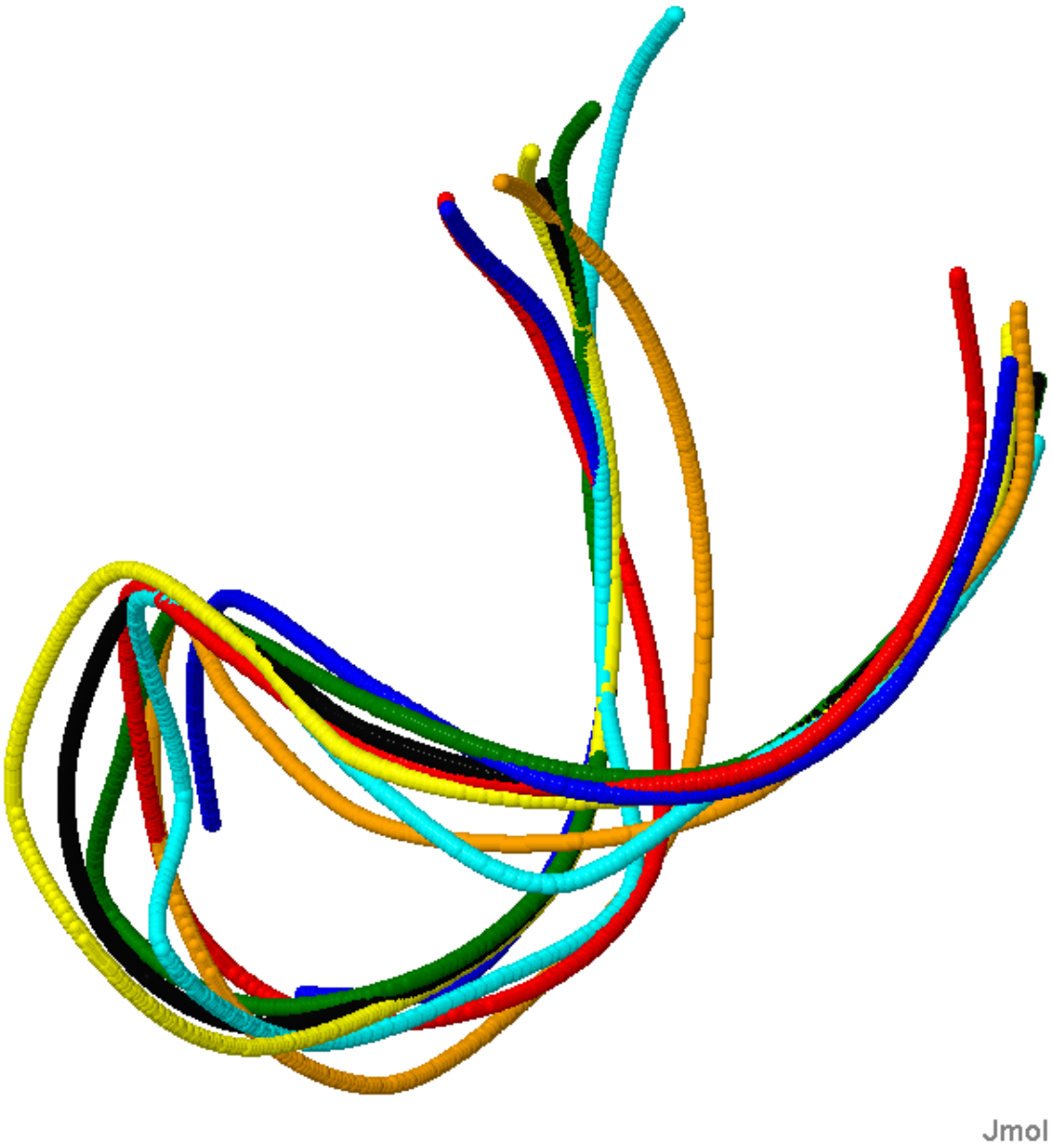


13.2.2 Family 2

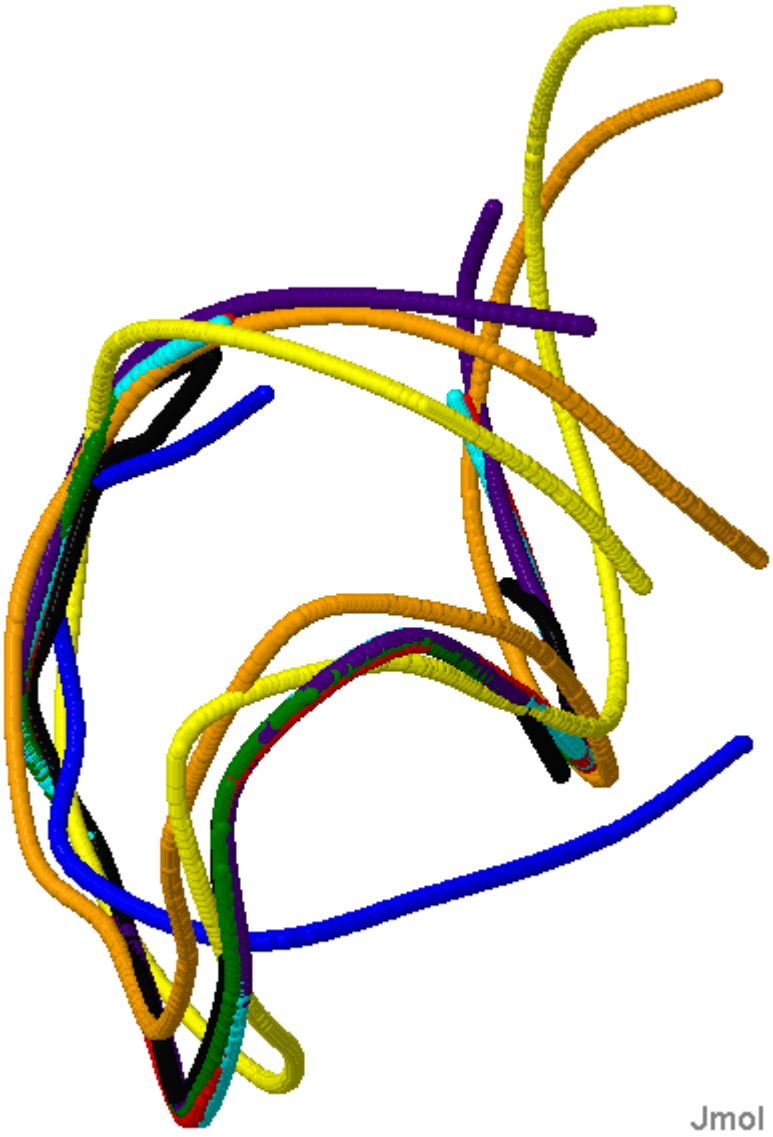
13.2.3 Family 3



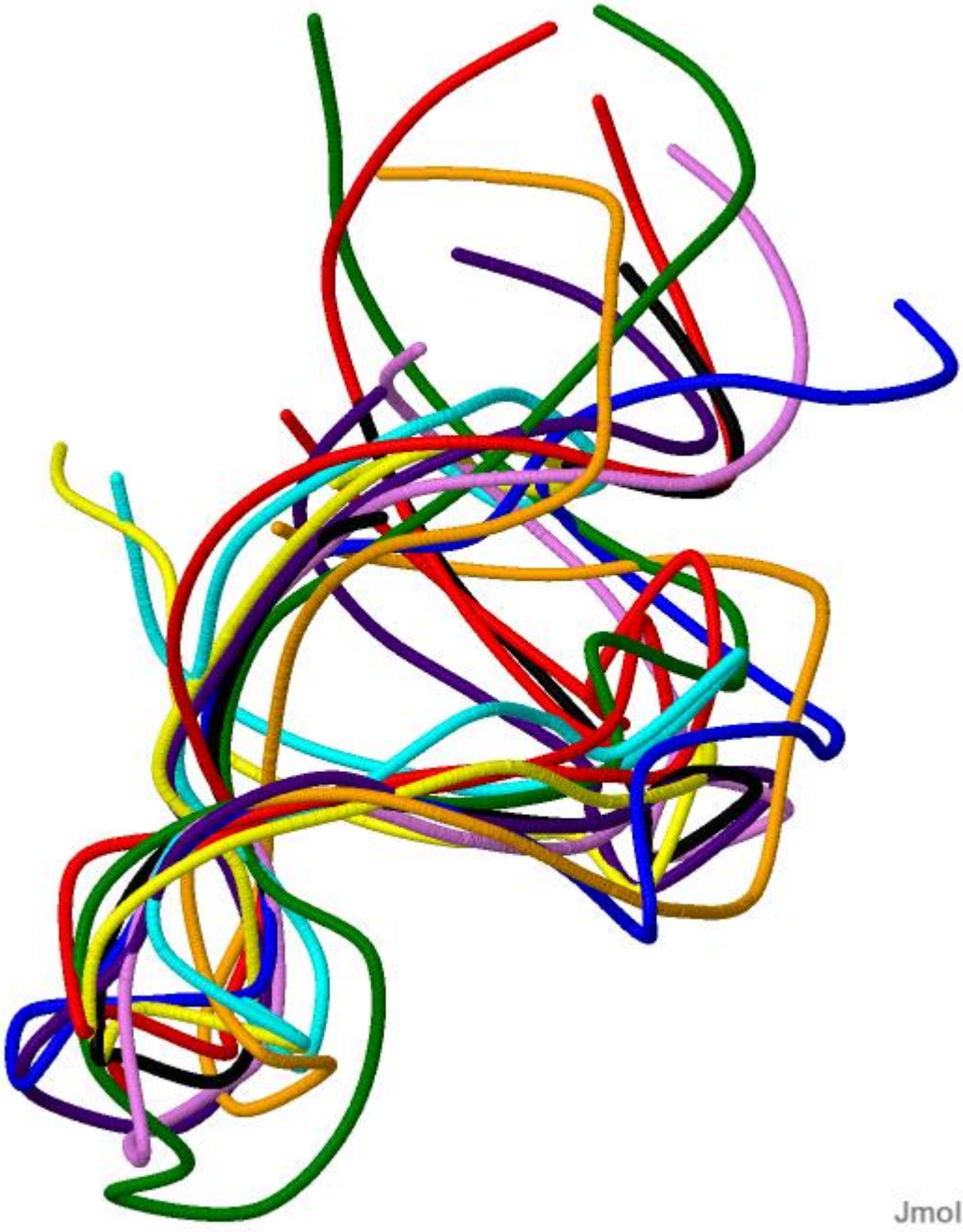
13.2.4 Family 6



13.2.5 Family 10



13.2.6 Family 11



13.2.7 Family 13